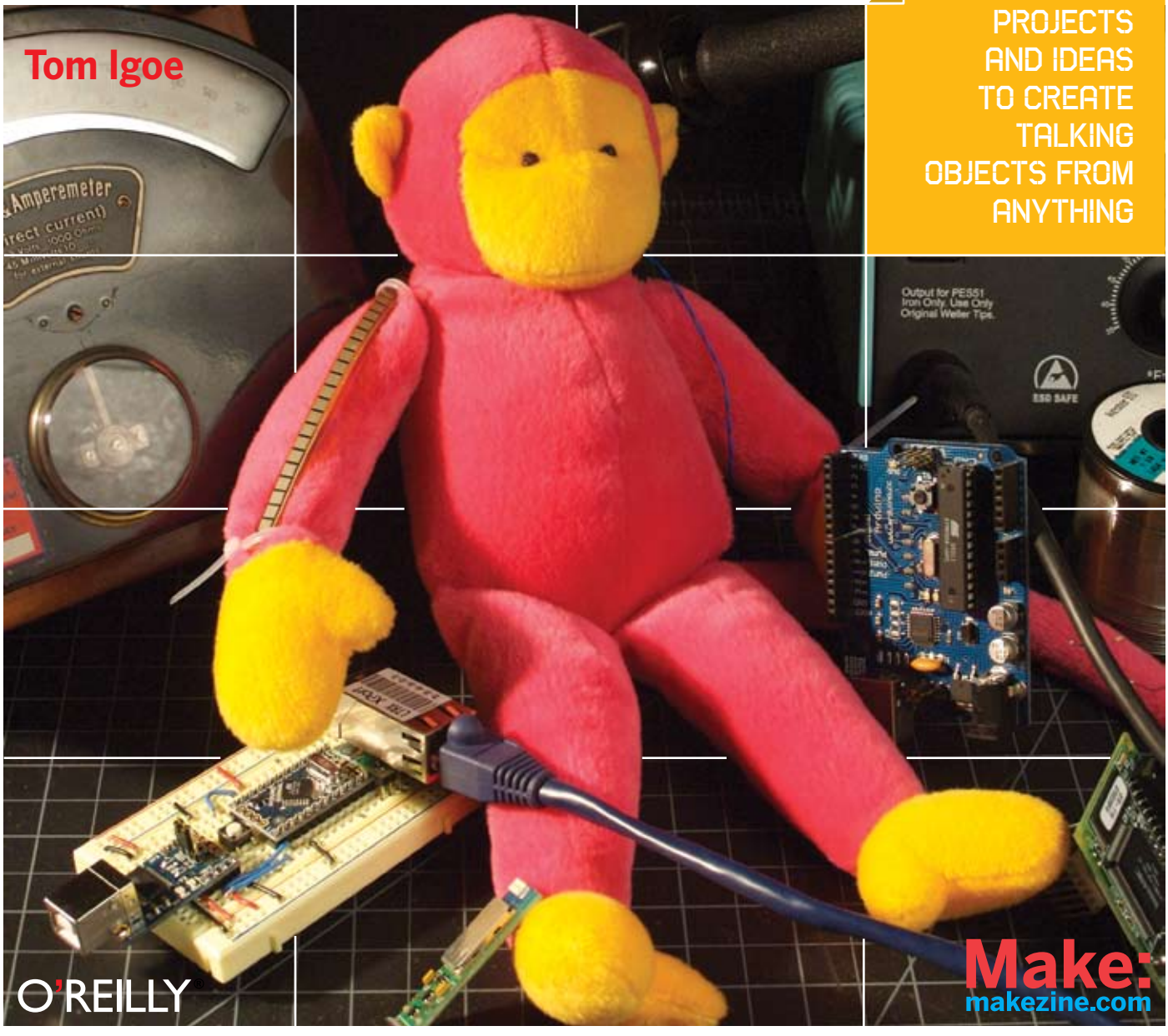
Make: PROJECTS

Making Things Talk

Practical Methods for Connecting Physical Objects

Tom Igoe

PROJECTS AND IDEAS TO CREATE TALKING OBJECTS FROM ANYTHING



O'REILLY

Make: makezine.com

Making Things Talk

by Tom Igoe

Copyright © 2007 O'Reilly Media, Inc. All rights reserved. Printed in U.S.A.

Published by Make:Books, an imprint of Maker Media, a division of O'Reilly Media, Inc.
1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use.
For more information, contact our corporate/institutional sales department:
800-998-9938 or *corporate@oreilly.com*.

Print History
September 2007
First Edition

Publisher: Dale Dougherty
Associate Publisher and Executive Editor: Dan Woods
Editor: Brian Jepson
Copy Editor: Nancy Kotary
Creative Director: Daniel Carter
Designer: Katie Wilson
Production Manager: Terry Bronson
Indexer: Patti Schiendelman
Cover Photograph: Tom Igoe

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. The MAKE: Projects series designations, Making Things Talk, and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc. was aware of the trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

Please note: Technology, and the laws and limitations imposed by manufacturers and content owners, are constantly changing. Thus, some of the projects described may not work, may be inconsistent with current laws or user agreements, or may damage or adversely affect some equipment.

Your safety is your own responsibility, including proper use of equipment and safety gear, and determining whether you have adequate skill and experience. Power tools, electricity, and other resources used for these projects are dangerous unless used properly and with adequate precautions, including safety gear. Some illustrative photos do not depict safety precautions or equipment, in order to show the project steps more clearly. These projects are not intended for use by children.

Use of the instructions and suggestions in Making Things Talk is at your own risk. O'Reilly Media, Inc., disclaims all responsibility for any resulting damage, injury, or expense. It is your responsibility to make sure that your activities comply with applicable laws, including copyright.

ISBN-10: 0-596-51051-9
ISBN-13: 978-0-596-51051-0

This excerpt is protected by copyright law. It is your responsibility to obtain permissions necessary for any proposed use of this material. Please direct your inquiries to permissions@oreilly.com.

FOCUS
GAIN

1

MAKE: PROJECTS

The Tools

This book is a cookbook of sorts, and this chapter covers the staple ingredients. The concepts and tools you'll use in every chapter are introduced here. There's enough information on each tool to get you to the point where you can make the tool say **"Hello World!"** Chances are you've used some of the tools in this chapter before, or other tools just like them. Skip past the things you know and jump into learning the tools that are new to you. You may want to explore some of the less-familiar tools on your own to get a sense of what they can do. The projects in the following chapters only scratch the surface of what's possible for most of these tools. References for further investigation are provided.

Happy Feedback Machine by Tuan Anh T. Nguyen
The main pleasure of interacting with this piece comes from the feel of flipping the switches and turning the knobs. The lights and sounds produced as a result are secondary, and most people who play with it remember the feel of it rather than its behavior.

It Starts with the Stuff You Touch

All of the objects that you'll encounter in this book, tangible or intangible, will have certain behaviors. Software objects will send and receive messages, store data, or both. Physical objects will move, light up, or make noise. The first question to ask about any of them is: what does it do? The second is: how do I make it do what it's supposed to do? Or, more simply, what is its interface?

An object's interface is made up of three elements. First, there's the physical interface. This is the stuff you touch. The knobs, switches, keys, and other sensors that make up the physical interface react to your actions. The connectors that join objects are also part of the physical interface. Many of the projects in this book will show you how to build physical interfaces. Every network of objects begins and ends with a physical interface. Even though some objects in a network (software objects) have no physical interface, people build their mental models of how a system works based on the physical interface. A computer is much more than the keyboard, mouse, and screen, but that's what we think of it as, because that's what we see and touch. You can build all kinds of wonderful functions into your system, but if those functions aren't apparent in the things people get to see, hear, and touch, your wonderful functions will never get used. Remember the lesson of the VCR clock that constantly blinks 12:00 because no one can be bothered to learn how to set it: if the physical interface isn't good, the rest of the system suffers.

Second, there's the software interface, the commands that you send to the object to make it respond. In some projects, you'll invent your own software interface, and in others, you'll rely on existing interfaces to do the work for you. The best software interfaces have simple, consistent functions that result in predictable outputs. Unfortunately, not all software interfaces are as simple as you'd like them to be, so be prepared to have to experiment a little to get some software objects to do what you think they should do. When you're learning a new software interface, it helps to approach it mentally in the same way you do with a physical interface. Don't try to use all the functions at once. Learn what each function does on its own before you try to use them all together. You don't learn to play the piano by starting with a Bach fugue — you start one note at a time. Likewise, you don't learn a software interface by writing a full application with it — you learn it one function at a time. There are many projects in this book; if you find any of their software functions confusing, write a simple program that demonstrates just that function, then return to the project.

Finally, there's the electrical interface, the pulses of electrical energy sent from one device to another to be interpreted as information. Unless you're designing new objects or the connections between them, you never have to deal with this interface. When you're designing new objects or the networks that connect them, however, you have to know and understand a few things about the electrical interface, so that you know how to match up objects that might have slight differences in their electrical interfaces.
X

It's About Pulses

In order to communicate with each other, objects use communications protocols. A protocol is a series of mutually agreed-upon standards for communication between two or more objects.

Serial protocols like RS-232, USB, and IEEE 1394 (also known as FireWire and i.Link) connect computers to printers, hard drives, keyboards, mice, and other peripheral devices. Network protocols like Ethernet and TCP/IP connect multiple computers to each other through network hubs, routers, and switches. A communications protocol usually defines the rate at which messages are exchanged, the arrangement of data in the messages, and the grammar of the exchange. If it's a protocol for physical objects, it will also specify the electrical characteristics, and sometimes even the physical shape of the connectors. Protocols don't specify what happens between objects, however. The commands to make an object do something rely on protocols in the same way that clear instructions rely on good grammar. You can't give good instructions if you can't form a good sentence.

One thing that all communications protocols share, from the simplest chip-to-chip message to the most complex network architecture, is this: it's all about pulses of energy. Digital devices exchange information by sending timed pulses of energy across a shared connection. The USB connection from your mouse to your computer uses two wires for transmission and reception, sending timed pulses of electrical energy across those wires. Likewise, wired network connections are made up of timed pulses of electrical energy sent down the wires. For longer distances and higher bandwidth, the electrical wires may be replaced with fiber optic cables carrying timed pulses of light. In cases where a physical connection is inconvenient or impossible, the transmission can be sent using pulses of radio energy between radio transceivers (a transceiver is two-way radio, capable of transmitting and receiving). The meaning of data pulses is independent of the medium that's carrying them. You can use the same sequence of pulses whether you're sending them across wires, fiber optic cables, or radios. If you keep in mind that all of the communication you're dealing with starts with a series of pulses, and that somewhere there's a guide explaining the sequence of those pulses, you can work with any communication system you come across.
X

Computers of all Shapes and Sizes

You'll encounter at least four different types of computers in this book, grouped according to their physical interfaces. The most familiar of these is the personal computer. Whether it's a desktop or a laptop machine, it's got a keyboard, a screen, and a mouse, and you probably use it just about every working day. These three elements: the keyboard, the screen, and the mouse — make up its physical interface.

The second type of computer you'll encounter in this book, the microcontroller, has no physical interface that humans can interact with directly. It's just an electronic chip with input and output pins that can send or receive electrical pulses. Using a microcontroller is a three-stage process:

1. You connect sensors to the inputs to convert physical energy like motion, heat, and sound into electrical energy.
2. You attach motors, speakers, and other devices to the outputs to convert electrical energy into physical action.
3. Finally, you write a program to determine how the input changes affect the outputs.

In other words, the microcontroller's physical interface is whatever you make of it.

The third type of computer in this book, the network server, is basically the same as a desktop computer, and may even have a keyboard, screen, and mouse. Even though it can do all the things you expect of a personal computer, its primary function is to send and receive data over a network. Most people using servers don't think of them as physical things, because they only interact with them over a network, using their local computers as physical interfaces to the server. A server's most important interface for most users' purposes is its software interface.

The fourth group of computers is a mixed bag: mobile phones, music synthesizers, and motor controllers, to name a few. Some of them will have fully developed physical interfaces, some of them will have minimal physical interfaces but detailed software interfaces, and most will have a little of both. Even though you don't normally think of these devices as computers, they are. When you think of them as programmable objects, with interfaces that you can manipulate, it's easier to figure out how they can all communicate with each other, regardless of their end function.

X

Good Habits

Networking objects is a bit like love. The fundamental problem in both is that when you're sending a message, you never really know whether the receiver understands what you're saying, and there are a thousand ways for your message to get lost or garbled in transmission.

You may know why you feel the way you do, but your partner doesn't. All he or she has to go on are the words you say and the actions you take. Likewise, you may know exactly what message your local computer is sending, how it's sending it, and what all the bits mean, but the remote computer has no idea what they mean unless you program it to understand them. All it has to go on are the bits it receives. If you want reliable, clear communications (in love or networking), there are a few simple things you have to do:

- Listen more than you speak.
- Never assume that what you said is what they heard.
- Agree on how you're going to say things in advance.
- Ask politely for clarification when messages aren't clear.

Listen More Than You Speak

The best way to make a good first impression, and to maintain a good relationship, is to be a good listener. Listening is more difficult than speaking. You can speak anytime you want to, but you never know when the other person is going to say something, so you have to listen all the time. In networking terms, this means that you should write your programs such that they're listening for new messages most of the time, and sending messages only when necessary. It's often easier to send out messages all the time rather than figure out when it's appropriate, but it can lead to all kinds of problems. It usually doesn't take a lot of work to limit your sending, and the benefits far outweigh the costs.

Never Assume

What you say is not always what the other person hears. Sometimes it's a matter of misinterpretation, and other times, you may not have been heard clearly. If you assume that the message got through and continue on obliviously, you're in for a world of hurt. Likewise, you may be tempted to work out all the logic of your system, and all the steps of your messages before you start to connect things together, then build it, then test it all at once. Avoid that temptation.

It's good to plan the whole system out in advance, but build it and test it in baby steps. Most of the errors that occur in building these projects occur in the communication between objects. Always send a quick "Hello World!" message from one object to the others and make sure that the message got there intact before you proceed to the more complex details. Keep that "Hello World!" example on hand for testing when communication fails.

Getting the message wrong isn't the only wrong step you can make. Most of the projects in this book involve building the physical, software, and electrical elements of the interface. One of the most common mistakes people make when developing hybrid projects like these is to assume that the problems are all in one place. Quite often, I've sweated over a bug in the software transmission of a message, only to find out later that the receiving device wasn't even connected, or wasn't ready to receive messages. Don't assume that communication errors are in the element of the system with which you're most familiar.

They're most often in the element with which you're least familiar, and therefore are avoiding. When you can't get a message through, think about every link in the chain from sender to receiver, and check every one. Then check the links you overlooked.

Agree on How You Say Things

In good relationships, you develop a shared language based on shared experience. You learn the best ways to say things so that your partner will be most receptive, and you develop shorthand for expressing things that you repeat all the time. Good data communications also rely on shared ways of saying things, or protocols. Sometimes you make up a protocol yourself for all the objects in your system, and other times you have to rely on existing protocols. If you're working with a previously established protocol, make sure you understand what all the parts are before you start trying to interpret it. If you have the luxury of making up your own protocol, make sure you've considered the needs of both the sender and receiver when you define it. For example, you might decide to use a protocol that's easy to program on your web server, but turns out to be impossible to handle on your microcontroller. A little thought to the strengths and weaknesses on both sides of the transmission and a little compromise before you start to build will make things flow much more smoothly.

Ask Politely for Clarification

Messages get garbled in countless ways. Sometimes you hear one thing; it may not make much sense, but you act on it ... only to find out that your partner said something entirely different from what you thought. It's always best to ask nicely for clarification to avoid making a stupid mistake. Likewise, in network communications, it's wise to check that any messages you receive make sense. When they don't, ask for a repeat transmission. It's also wise to check that a message was sent, rather than assume. Saying nothing can be worse than saying something wrong. Minor problems can become major when no one speaks up to acknowledge that there's a problem. The same thing can occur in network communications. One device may wait forever for a message from the other side, not knowing that the remote device is unplugged, or perhaps it didn't get the initial message. When no response is forthcoming, send another message. Don't resend it too often, and give the other party time to reply before resending. Acknowledging messages may seem like a luxury, but it can save a whole lot of time and energy when you're building a complex system.

X

Tools

As you'll be working with the physical, software, and electrical interfaces of objects, the tools you'll need are physical tools, software, and (computer) hardware.

Physical Tools

If you've worked with electronics or microcontrollers before, chances are you have your own hand tools already. Figure 1-1 shows the ones used most frequently in this book. They're common tools, and can be obtained from many vendors. A few are listed in Table 1-1.

In addition to hand tools, there are some common electronic components that you'll use all the time. They're listed as well, with part numbers from the retailers featured most frequently in this book. Not all retailers will carry all parts, so there are many gaps in the table.

NOTE: **You'll find a number of component suppliers in this book. I buy from different vendors depending on who's got the best and the least expensive version of each part. Sometimes it's easier to buy from a vendor that you know carries what you need rather than search through the massive catalog of a vendor who might carry it cheaper. Feel free to substitute your favorite vendors. A list of vendors can be found in Appendix B.**

Figure 1-1. See list at right for number references.

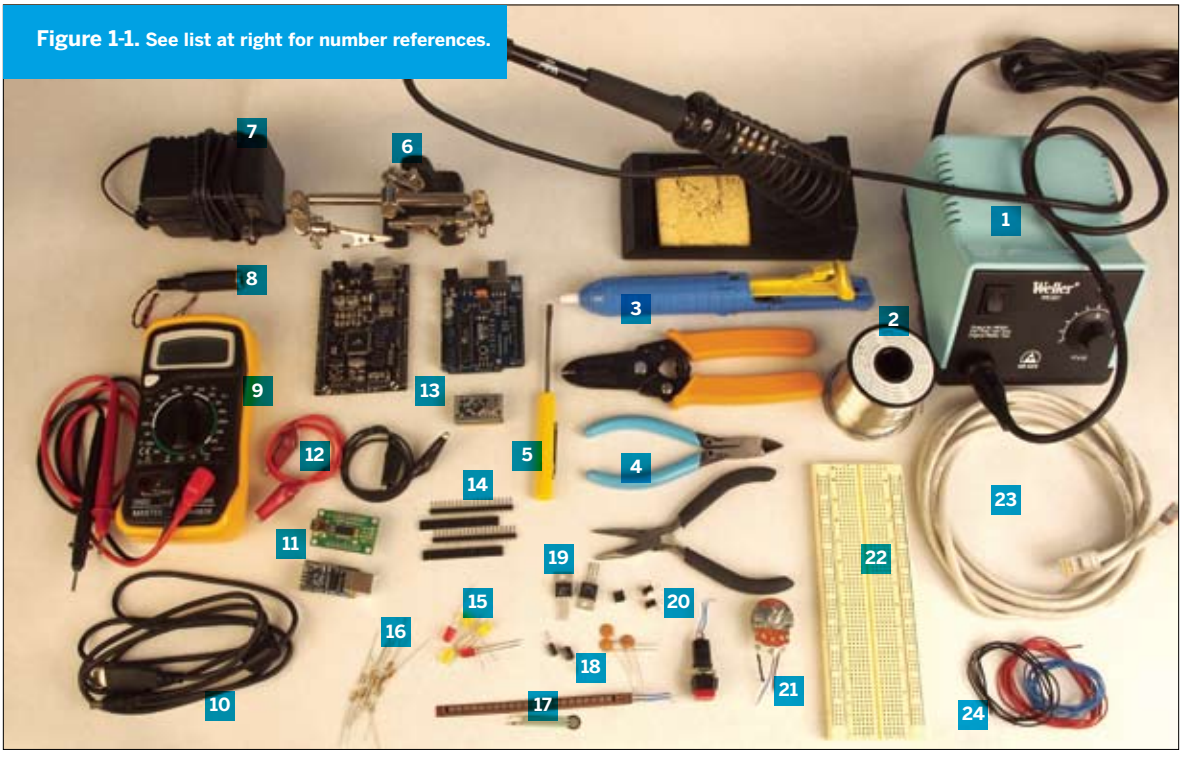


Table 1-1. Common tools for electronic and microcontroller work.

D Digi-Key (digikey.com)
I Images SI (imagesco.com)
J Jameco (jameco.com)
S SparkFun Electronics (sparkfun.com)

RESISTORS

100Ω	**D** 100QBK-ND, **J** 690620
220Ω	**D** 220QBK-ND, **J** 690700
470Ω	**D** 470QBK-ND, **J** 690785
1K	**D** 1.0KQBK, **J** 29663
10K	**D** 10KQBK-ND, **J** 29911
22K	**D** 22KQBK-ND, **J** 30453
100K	**D** 100KQBK-ND, **J** 29997
1M	**D** 1.0MQBK-ND, **J** 29698

CAPACITORS

0.1µF ceramic	**D** 399-4151-ND, **J** 15270
1µF electrolytic	**D** P10312-ND, **J** 94161
10µF electrolytic	**D** P11212-ND, **J** 29891, **S** COM-00523
100µF electrolytic	**D** P10269-ND, **J** 158394, **S** COM-00096

VOLTAGE REGULATORS

3.3V	**D** 576-1134-ND, **J** 242115, **S** COM-00526
5V	**D** LM7805CT-ND, **J** 51262, **S** COM-00107

ANALOG SENSORS

Flex sensors	**J** 150551, **I** FLX-01
FSRs	**P** 30056, **I** FSR-400, 402, 406, 408

LED

T1, Green clear	**D** 160-1144-ND, **J** 34761
T1, Red, clear	**D** 160-1665-ND, **J** 94511

TRANSISTORS

2N2222A	**J** 38236
TIP120	**J** 32993

DIODES

1N4004-R	**D** 1N4004-E3 or 23GI-ND, **J** 35992
3.3V zener (1N5226)	**D** 1N5226B-TPCT-ND, **J** 743488

PUSHBUTTONS

PCB	**D** SW400-ND, **J** 119011, **S** COM-00097
Panel Mount	**D** GH1344-ND, **J** 164559PS

SOLDERLESS BREADBOARDS

various	**D** 438-1045-ND, **J** 20723, 20600, **S** PRT-00137

HOOKUP WIRE

red	**J** 36856, **S** PRT-08023
black	**J** 36792, **S** PRT-08022
blue	**J** 36767
yellow	**S** PRT-08024

POTENTIOMETER

10K	**D** 29081

HEADER PINS

straight	**D** A26509-20-ND, **J** 103377, **S** PRT-00116
right angle	**D** S1121E-36-ND, **S** PRT-00553

HEADERS

female	**S** PRT-00115

BATTERY SNAP

9V	**D** 2238K-ND, **J** 101470PS, **S** PRT-00091

Handy hand tools for networking objects.

1 Soldering iron Middle-of-the-line is best here. Cheap soldering irons die fast, but a mid-range iron like the Weller WLC-100 work great for small electronic work. Avoid the Cold Solder irons. They solder by creating a spark, and that spark can damage static-sensitive parts like microcontrollers. Jameco (jameco.com): 146595; RadioShack: 640-2802 and 640-2078

2 Solder 21-23 AWG solder is best. Get lead-free solder if you can, it's healthier for you. Jameco: 668271; RadioShack: 640-0013

3 Desoldering pump This helps when you mess up while soldering. Jameco: 305226; SparkFun (sparkfun.com): TOL-00082

4 Wire stripper, Diagonal cutter, Needlenose pliers Avoid the 3-in-1 versions of these tools. They'll only make you grumpy. These three tools are essential for working with wire, and you don't need expensive ones to have good ones.
Wire stripper: Jameco: 159291; RadioShack: 640-2129A; SparkFun: TOL-00089
Diagonal cutter: Jameco: 161411; RadioShack: 640-2043; SparkFun: TOL-00070
Needlenose pliers: Jameco: 35473; RadioShack: 640-2033; SparkFun: TOL-00079

5 Mini-screwdriver Get one with both Phillips and slotted heads. You'll use it all the time. Jameco: 127271; RadioShack: 640-1963

6 Helping hands These make soldering much easier. Jameco: 681002

7 9–12V DC power supply You'll use this all the time, and you've probably got a spare from some dead electronic device. Make sure you know the polarity of the plug so you don't reverse polarity on a component and blow it up! Most of the devices shown in this book have a DC power jack that accepts a 2.1mm inner diameter/5.5mm outer diameter plug, so look for an adaptor with the same dimensions. Jameco: 170245 (12V, 1000mA); RadioShack: 273-1667 (3–12V, 800mA); SparkFun: TOL-00298

8 Power connector, 2.1mm inside diameter/5.5mm outside diameter You'll need this to connect your microcontroller module or breadboard to a DC power supply. This size connector is the most common for the power supplies that will work with the circuits you'll be building here. Jameco: 159610; Digi-Key (digikey.com): CP-024A-ND

9 Multimeter You don't need an expensive one. As long as it measures voltage, resistance, amperage, and continuity, it'll do the job. Jameco: 220812; RadioShack: 22-810; SparkFun: TOL-00078

10 USB cables You'll need both USB A-to-B (the most common USB cables) and USB A-to-mini-B (the kind that's common with digital cameras) for the projects in this book. SparkFun: CAB-00512, CAB-00598

11 Serial-to-USB converter This converter lets you speak TTL serial from a USB port. Breadboard serial-to-USB modules like the FT232 modules shown here are cheaper than the consumer models, and easier to use in the projects in this book. SparkFun: BOB-00718 or DEV-08165

12 Alligator clip test leads It's often hard to juggle the five or six things you have to hold when metering a circuit. Clip leads make this much easier. Jameco: 10444; RadioShack: 278-016; SparkFun: CAB-00501

13 Microcontroller module The microcontrollers shown here are the Arduino NG and the Arduino Mini. Available from SparkFun and Make (store.makezine.com) in the U.S., PCB-Europe in Europe (pcb-europe.net/catalog/) and from multiple distributors internationally. See arduino.cc/en/Main/Buy for details in your region.

14 Header pins You'll use these all the time. It's handy to have female ones around as well. Jameco: 103377; Digi-Key: A26509-20-ND; SparkFun: PRT-00116

15 Spare LEDs for tracing signals LEDs are to the hardware developer what print statements are to the software developer. They let you see quickly if there's voltage between two points, or if a signal's going through. Keep spares on hand. Jameco: 3476; RadioShack: 276-0069; Digi-Key: 160-1144-ND, 160-1665-ND

16 Resistors You'll need resistors of various values for your projects. Common values are listed in Table 1-1.

17 Analog sensors (variable resistors) There are countless varieties of variable resistors to measure all kinds of physical properties. They're the simplest of analog sensors, and they're very easy to build into test circuits. Flex sensors and force-sensing resistors are handy for testing a circuit or a program.
Flex sensors: Jameco: 150551; Images SI: FLX-01
Force-sensing resistors: Parallax: 30056; Images SI: FSR-400, 402, 406, 408

18 Capacitors You'll need capacitors of various values for your projects. Common values are listed in Table 1-1.

19 Voltage regulators Voltage regulators take a variable input voltage and output a constant (lower) voltage. The two most common you'll need for these projects are 5V and 3.3V. Be careful when using a regulator that you've never used before. Check the data sheet to make sure you have the pin connections correct.
3.3V: Digkey: 576-1134-ND; Jameco: 242115; SparkFun: COM-00526
5V: Digkey: LM7805CT-ND; Jameco: 51262; SparkFun: COM-00107

20 Pushbuttons There are two types you'll find handy: the PCB-mount type like the ones you find on Wiring and Arduino boards, used here mostly as reset buttons for breadboard projects; and panel-mount types used for interface controls for end users. But you can use just about any type you want.
PCB-mount type: Digi-Key: SW400- ND; Jameco: 119011; SparkFun: COM-00097
Panel-mount type: Digi-Key: GH1344-ND; Jameco: 164559PS

21 Potentiometers You'll need potentiometers to let people adjust settings in your project. Jameco: 29081

22 Solderless breadboard Having a few around can be handy. I like the ones with two long rows on either side, so you can run power and ground on both sides. Jameco: 20723 (2 bus rows per side); RadioShack: 276-174 (1 bus row per side); Digi-Key: 438-1045-ND; SparkFun: PRT-00137

23 Ethernet cables A couple of these will come in handy. Jameco: 522781

24 Black, red, blue, yellow wire 22 AWG solid-core hook-up wire is best for making solderless breadboard connections. Get at least three colors, and always use red for voltage and black for ground. A little organization of your wires can go a long way.
Black: Jameco: 36792
Blue: Jameco: 36767
Green: Jameco: 36821
Red: Jameco: 36856; RadioShack: 278-1215
Yellow: Jameco: 36919
Mixed: RadioShack: 276-173

Figure 1-2
The Processing editor window.

Software Tools

Processing

The multimedia programming environment used in this book is called Processing. It's based on Java, and made for designers, artists, and others who don't need to know all the gory details of programming, but want to get something done. It's a useful tool for explaining programming ideas because it takes relatively little Processing code to make big things happen, such as opening a network connection, connecting to an external device through a serial port, or controlling a camera through FireWire. It's a free, open source tool available from www.processing.org. Because it's based on Java, you can include Java classes and methods in your Processing programs. It runs on Mac OS X, Windows, and Linux, so almost anyone can run Processing on their favorite operating system. If you don't like working in Processing, you should be able to use the code samples here and their comments as pseudocode for whatever multimedia environment you prefer. Once you've downloaded and installed Processing on your computer, open the application. You'll get a screen that looks like Figure 1-2.

» Here's your first Processing program. Type this into the editor window, and press the Run button on the top left-hand side of the toolbar:

```
println("Hello World!\n");
```

It's not too flashy a program, but it's a classic. It should print Hello World! in the message box at the bottom of the editor window. It's that easy.

Programs in Processing are called sketches, and all the data for a sketch is saved in a folder with the sketch's name. The editor is very basic, without a lot of clutter to get in your way. The toolbar has buttons to run and stop a sketch, create a new file, open an existing sketch, save the current sketch, or export to a Java applet. You can also export your sketch as a standalone application from the File menu. Files are normally stored in a subdirectory of your **Documents** folder called **Processing**, but you can save them wherever you prefer if you don't like them there.

Here's a second program that's a bit more exciting. It illustrates some of the main programming structures in Processing:

```
/*
    Triangle drawing program
    Language: Processing

    Draws a triangle whenever the mouse button is not pressed.
    Erases when the mouse button is pressed.

*/

// declare your variables:
float redValue = 0;    // variable to hold the red color
float greenValue = 0;  // variable to hold the green color
float blueValue = 0;   // variable to hold the blue color

// the setup() method runs once at the beginning of the program:

void setup() {
  size(320, 240);    // sets the size of the applet window
  background(0);     // sets the background of the window to black
  fill(0);           // sets the color to fill shapes with (0 = black)
  smooth();          // draw with antialiased edges
}

// the draw() method runs repeatedly, as long as the applet window
// is open.  It refreshes the window, and anything else you program
// it to do:

void draw() {

  // Pick random colors for red, green, and blue:
  redValue = random(255);
  greenValue = random(255);
  blueValue = random(255);

  // set the line color:
  stroke(redValue, greenValue, blueValue);

  // draw when the mouse is up (to hell with conventions):
  if (mousePressed == false) {
    // draw a triangle:
    triangle(mouseX, mouseY, width/2, height/2,pmouseX, pmouseY);
  }
  // erase when the mouse is down:
  else {
    background(0);
    fill(0);
  }
}
```

Every Processing program has two main routines, setup() and draw(). setup() happens once at the beginning of the program. It's where you set all your initial conditions, like the size of the applet window, initial states for variables, and so forth. draw() is the main loop of the program. It repeats continuously until you close the applet window.

In order to use variables in Processing, you have to declare the variable's data type. In the preceding program, the variables redValue, greenValue, and blueValue are all float types, meaning that they're floating decimal-point numbers. Other common variable types you'll use are ints (integers), booleans (true or false values), Strings of text, and bytes.

Like C, Java and many other languages, Processing uses C-style syntax. All functions have a data type, just like variables (and many of them are the void type, meaning that they don't return any values). All lines end with a semicolon, and all blocks of code are wrapped in curly brackets. Conditional statements (if-then statements), for-next loops, and comments all use the C syntax as well. The preceding code illustrates all of these except the for-next loop.

» Here's a typical for-next loop. Try this in a sketch of its own (to start a new sketch, select New from Processing's File menu):

```
for (int myCounter = 0; myCounter <=10; myCounter++) {
  println(myCounter);
}
```

Processing is a fun language to play with, because you can make interactive graphics very quickly. It's also a simple introduction to Java for beginning programmers. If you're a Java programmer already, you can include Java directly in your Processing programs. Processing is expandable through code libraries. You'll be using two of the Processing code libraries frequently in this book: the serial library and the networking library.

For more on the syntax of Processing, see the language reference guide at www.processing.org. To learn more about programming in Processing, check out *Processing: A Programming Handbook for Visual Designers and Artists*, by Casey Reas and Ben Fry (MIT Press, 2007), the creators of Processing.

BASIC users: If you've never used a C-style for-next loop, it can seem a bit forbidding. What this bit of code does is establish a variable called myCounter**. As long as number is less than or equal to ten, it executes the instructions in the curly brackets.** myCounter++ **tells the program to add one to** myCounter **each time through the loop. The equivalent BASIC code is:**

```
for myCounter = 0 to 10
  Print myCounter
next
```

Mac OS X Users: Once you've downloaded and installed Processing, there's an extra step you'll need to take that will make the projects in this book that use Processing possible. Go to the Processing application directory, then to the libraries/serial/ subdirectory. There's a file there called macosx_setup.command. Double-click this. It will run a script that enables Processing to use serial communication to USB, Bluetooth, and other devices. A terminal window will open and run a script that will ask you a few questions. It will also ask for your administrator password, so don't run it unless you have administrator access to your machine. Say "yes" to anything it asks, and provide your password when needed. When it's done, you'll be able to use the serial ports of your computer through Processing. You'll be making heavy use of this capability later on in this book.

Remote Access Applications

One of the most effective debugging tools you'll use in making the projects in this book is a command-line remote access program, which allows you access to the command-line interface of a remote computer. If you've never used a command-line interface before, you'll find it a bit awkward at first, but you get used to it pretty quickly. This tool is especially important when you need to log into a web server, as you'll need the command line to create PHP scripts that will be used in this book.

Most web hosting providers are based on Linux, BSD, Solaris or some other Unix-like operating system. So, when you need to do some work on your web server, you may need to make a command-line connection to your web server.

If you already know how to create PHP and HTML documents and upload them to your web server, you can skip ahead to the "PHP" section.

In a command-line interface, everything is done by typing commands at the cursor. The programs you'll be running and the files you'll be writing and reading aren't on your machine. When you're using the PHP programming language described shortly, for example, you'll be using programs and reading files directly on the web host's computer.

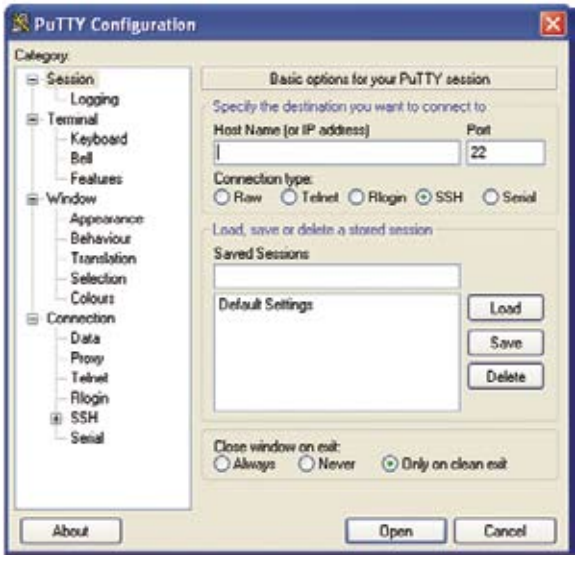
Although this is the most direct way to work with PHP, some people prefer to work more indirectly, by writing text files on their local computers and uploading them to the remote computer. Depending on how restrictive your web hosting service is, this may be your only option (however, there are many inexpensive hosting companies that offer full command-line access). Even if you prefer to work this way, there are times in this book when the command line is your only option, so it's worth getting to know a little bit about it now.

On Windows computers, there are a few remote access programs available, but the one that you'll use here is called PuTTY. You can download it from www.puttyssh.org. Download the Windows-style installer and run it. On Mac OS X and Linux, you can use OpenSSH, which is included with both operating systems, and can be run in the Terminal program with the command ssh.

Before you can run OpenSSH, you'll need to launch a terminal emulation program, which gives you access to your Linux or Mac OS X command line. On Mac OS X, the program is called Terminal, and you can find it in the **Utilities** subdirectory of the **Applications** directory. On Linux, look for a program called xterm, rxvt, Terminal, or Konsole.

NOTE: ssh **is a more modern cousin of a longtime Unix remote access program called** telnet**.** ssh **is more secure, in that it scrambles all data sent from one computer to another before sending it, so it can't be snooped on en route.** telnet **sends all data from one computer to another with no encryption. You should use** ssh **to connect from one machine to another whenever you can. Where** telnet **is used in this book, it's because it's the only tool that will do what's needed for the examples in question. Think of** telnet **as an old friend: maybe not the coolest guy on the block, maybe he's a bit of a gossip, but he's stood by you forever, and you know you can trust him to do the job when everyone else lets you down.**

X



Making the SSH Connection

Mac OS X and Linux

Open your terminal program. These Terminal applications give you a plain text window with a greeting like this:

```
Last login: Wed Feb 22 07:20:34 on ttyp1
ComputerName:~ username$
```

Type ssh username@myhost.com at the command line to connect to your web host. Replace username and myhost.com with your username and host address.

Windows

On Windows, you'll need to start up PuTTY (see Figure 1-3). To get started, type myhost.com (your web host's name) in the Host Name field, choose the SSH protocol, and then click Open.

The computer will try to connect to the remote host, and asks for your password when it connects. Type it (you won't see what you type), followed by the Enter key.

Figure 1-3
The main PuTTY window.

Using the Command Line

Once you've connected to the remote web server, you should see something like this:

```
Last login: Wed Feb 22 08:50:04 2006 from 216.157.45.215
[userid@myhost ~]$
```

Now you're at the command prompt of your web host's computer, and any command you give will be executed on that computer. Start off by learning what directory you're in. To do this, type the following:

```
pwd
```

which stands for "print working directory." It asks the computer to list the name and pathname of the directory in which you're currently working. You'll see that many Unix commands are very terse, so you have to type less. The downside of this is that it makes them harder to remember. The server will respond with a directory path, such as:

```
/home/igoe
```

This is the home directory for your account. On many web servers, this directory contains a subdirectory called **public_html** or **www**, which is where your web files belong. Files that you place in your home directory (that is, outside or **www** or **public_html**) can't be seen by web visitors.

NOTE: You should check with your web host to learn how the files and directories in your home directory are set up.

To find out what files are in a given directory, use the list (ls) command, like so:

```
ls -l .
```

NOTE: The dot is shorthand for "the current working directory." Similarly, a double dot is shorthand for the directory (the parent directory) that contains the current directory.

The –l means "list long." You'll get a response like this:

```
total 44
drwxr-xr-x  13 igoe users 4096 Apr 14 11:42 public_html
drwxr-xr-x   3 igoe users 4096 Nov 25  2005 share
```

This is a list of all the files and subdirectories of the current working directories, and their attributes. The first column lists who's got permissions to do what (read, modify, or execute/run a file). The second lists how many links there are to that file elsewhere on the system; it's not something you'll have much need for, most of the time. The third column tells you who owns it, and the fourth tells you the group (a collection of users) the file belongs to. The fifth lists its size, and the sixth lists the date it was last modified. The final column lists the filename.

In a Unix environment, all files whose names begin with a dot are invisible. Some files, like access-control files that you'll see later in the book, need to be invisible. You can get a list of all the files, including the invisible ones, using the –a modifier for ls, this way:

```
ls -la
```

To move around from one directory to another, there's a "change directory" command, cd. To get into the **public_html** directory, for example, type:

```
cd public_html
```

To go back up one level in the directory structure, type:

```
cd ..
```

To return to your home directory, use the ~ symbol, which is shorthand for your home directory:

```
cd ~
```

If you type cd on a line by itself, it also takes you to your home directory.

If you wanted to go into a subdirectory of a directory, for example the **cgi-bin** directory inside the **public_html** directory, you'd type cd public_html/cgi-bin. You can type the absolute path from the main directory of the server (called the root) by placing a **/** at the beginning of the file's pathname. Any other file pathname is called a relative path.

To make a new directory, type:

```
mkdir directoryname
```

This command will make a new directory in the current working directory. If you then use ls -l to see a list of files in the working directory, you'll see a new line with the new directory. If you then type cd directoryname to switch to the new directory and ls –la to see all of its contents, you'll see only two listings:

```
drwxr-xr-x  2 tqi6023 users 4096 Feb 17 10:19 .
drwxr-xr-x  4 tqi6023 users 4096 Feb 17 10:19 ..
```

The first file, **.** , is a reference to this directory itself. The second, **..** , is a reference to the directory that contains it. Those two references will exist as long as the directory exists. You can't change them.

To remove a directory, type:

```
rmdir directoryname
```

You can remove only empty directories, so make sure that you've deleted all the files in a directory before you remove it. rmdir won't ask you if you're sure before it deletes your directory, though, so be careful. Don't remove any directories or files that you didn't make yourself until you know your way around.

Controlling Access to Files

Type `ls -l .` to get a list of files in your current directory and take a closer look at the permissions on the files. For example, a file marked drwx------ means that it's a directory, and that it's readable, writable, and executable by the system user that created the directory (also known as the owner of the file). Or take the file marked -rw-rw-rw. The – at the beginning means it's a regular file, not a directory, and that the owner, the group of users that the file belongs to (usually, this is the group that the owner is a member of), and everyone else who accesses the system can read and write to this file. The first rw- refers to the owner, the second refers to the group, and the third refers to the rest of the world. If you're the owner of a file, you can change its permissions using the chmod command:

```
chmod go –w filename
```

The options following chmod refer to which users you want to affect. In the preceding example, you're removing write permission (-w) for the group (g) that the file belongs to, and for all others (o) besides the owner of the file. To restore write permissions for the group and others, and to also give them execute permission, you'd type:

```
chmod go +wx filename
```

A combination of u for user, g for group, and o for others, and a combination of + and – and r for read, w for write, and x for execute gives you the capability to change permissions on your files for anyone on the system. Be careful not to accidentally remove permissions from yourself (the user). Also, get in the habit of not leaving files accessible to the group and others unless you need to: on large hosting providers, it's not unusual for you to be sharing a server with hundreds of other users!

Creating, Viewing, and Deleting Files

Two other command-line programs you'll find useful are nano and less. nano is a text editor. It's very bare-bones, and you may prefer to edit your files using your favorite text editor on your own computer and then upload them to your server. But for quick changes right on the server, nano is great. To make a new file, type:

```
nano filename.txt
```

The nano editor will open up. Figure 1-4 shows what it looked like after I typed in some text.

All the commands to work in nano are keyboard commands you type using the Control key. For example, to exit the program, type Control-X. The editor will then ask you if you want to save, and prompt you for a filename. The most common commands are listed along the bottom of the screen.

While nano is for creating and editing files, less is for reading them. less takes any file and displays it to the screen one screenful at a time. To see the file you just created in nano, for example, type:

```
less filename.txt
```

You'll get a list of the file's contents, with a : prompt at the bottom of the screen. Press the spacebar for the next screenful. When you've read enough, type q to quit. There's not much to less, but it's a handy way to read long files. You can even send other commands through less (or almost any command-line program) using the pipe (|) operator. For example, try this:

```
ls –la . | less
```

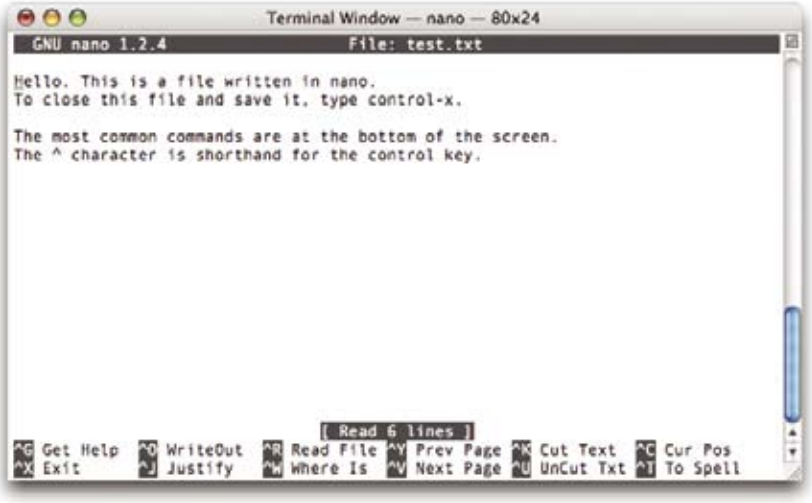

Terminal Window — nano — 80x24
GNU nano 1.2.4 File: test.txt
Hello. This is a file written in nano.
To close this file and save it, type control-x.
The most common commands are at the bottom of the screen.
The ^ character is shorthand for the control key.
[Read 6 lines]
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Txt ^T To Spell

Figure 1-4
The nano text editor.

Once you've created a file, you can delete it using the rm command, like this:

```
rm filename
```

Like rmdir, rm won't ask you if you're sure before it deletes your file, so use it carefully.

There are many other commands available in the Unix command shell, but these will suffice to get you started for now. For more information, type help at the command prompt to get a list of commonly used commands. For any command, you can get its user manual by typing man commandname. For more on getting around Unix and Linux systems using the command line, see *Learning the Unix Operating System* by Jerry Peek, John Strang, and Grace Todino-Gonguet. When you're ready to close the connection to your server, type: logout

PHP

The server programs in this book are mostly in PHP. PHP is one of the most common scripting languages for applications that run on the web server (server-side scripts). Server-side scripts are programs that allow you to do more with a web server than just serve fixed pages of text or HTML. They allow you to access databases through a browser, save data from a web session to a text file, send mail from a browser, and more. You'll need a web hosting account with an Internet service provider for most of the projects in this book, and it's likely that your host already provides access to PHP. If not, talk to your system administrator to see whether it can be installed.

To get started with PHP, you'll need to make a remote connection to your web hosting account using ssh as you did in the last section. Some of the more basic web hosts don't allow ssh connections, so check with yours to see whether they do (and if yours doesn't, look around for an inexpensive hosting company that does; it will be well worth it for the flexibility of working from the command line). Once you're connected, type: php -v

You should get a reply like this:

```
PHP 4.3.9 (cgi) (built: Nov  4 2005 11:49:43)
Copyright (c) 1997-2004 The PHP Group
Zend Engine v1.3.0, Copyright (c) 1998-2004 Zend
Technologies
```

This tells what version of PHP is installed on your server. The code in this book was written using PHP4, so as long as you're running that version or later, you'll be fine. PHP makes it easy to write web pages that can display results from databases, send messages to other servers, send email, and more.

Most of the time, you won't be executing your PHP scripts directly from the command line. Instead, you'll be calling the web server application on your server, most likely a program called Apache, and asking it for a file (this is all accomplished simply by opening a web browser, typing in the address of a document on your web server, and pressing Enter — just like visiting any other web page). If the file you ask for is a PHP script, the web server application will look for your file and execute it. It'll then send a message back to you with the results.

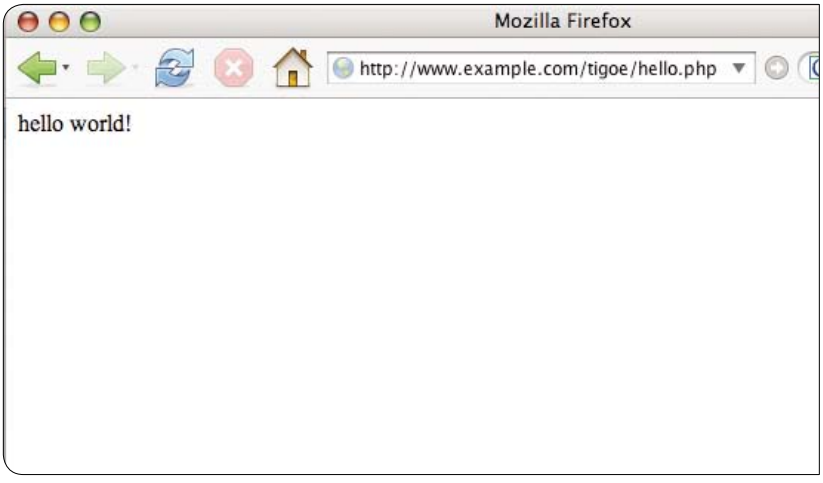


Figure 1-5
The results of your first PHP script, in a browser.

For more on this, see Chapter 3. For now, let's get a simple PHP program or two working. Here's your first PHP program. Open your favorite text editor, type this in, and save it on the server with the name **hello.php** in your **public_html** directory. (Your web pages may be stored in a different directory, such as **www** or **web/public**.)

```
<?php
echo "<html><head></head><body>\n";
echo "hello world!\n";
echo "</body></html>\n";
?>
```

Now, back at the command line, type the following to see the results:

```
php hello.php
```

You should get the following response:

```
<html><head></head><body>
hello world!
</body></html>
```

Now try opening this file in a browser. To see this program in action, open a web browser and navigate to the address of this file on your website. Because you saved it in **public_html,** the address is http://www.example.com/hello.php (replace www.example.com with your web site and any additional path info needed to access your home files, such as http://tigoe.net/~tigoe/hello.php). You should get a web page like the one in Figure 1-5.

If you see the PHP source code instead of what's shown in Figure 1-5, you may have opened up the PHP script as a local file (make sure your web browser's location bar says http:// **instead of** file://**).**

If it still doesn't work, your web server may not be configured or PHP. Another possibility is that your web server uses a different extension for php scripts, such as .php4. Consult with your web hosting provider for more information.

You may have noticed that the program is actually printing out HTML text. PHP was made to be combined with HTML. In fact, you can even embed PHP in HTML pages, by using the <? and ?> tags that start and end every PHP script. If you get an error when you try to open your PHP script in a browser, ask your system administrator if there are any requirements as to which directories PHP scripts need to be in on your server, or on the file permissions for your PHP scripts.

Here's a slightly more complex PHP script. Save it to your server in the **public_html** directory as **time.php**:

```
<?php
/*
   Date printer
   Language: PHP

   Prints the date and time in an HTML page.
*/
//    Get the date, and format it:
$date = date("Y-m-d h:i:s\t");

// print the beginning of an HTML page:
echo "<html><head></head><body>\n";
echo "hello world!<br>\n";
// Include the date:
echo "Today's date: $date<br>\n";
// finish the HTML:
echo "</body></html>\n";
?>
```

To see it in action, type http://www.example.com/time.php into your browser. You should get the date and time. You can see this program uses a variable, $date, and calls a built-in PHP function, date(), to fill the variable. You don't have to declare the types of your variables in PHP. Any simple, or scalar, variable begins with a $ and can contain an integer, a floating point number, or a string. PHP uses the same C-style syntax as Processing, so you'll see that if-then statements, repeat loops, and comments all look familiar.

For more on PHP, check out www.php.net, the main source for PHP, where you'll find some good tutorials on how to use it. You can also check out *Learning PHP 5* by David Sklar (O'Reilly Media, Inc., 2004) for a more in-depth treatment.

Serial Communication Tools

The remote access programs in the earlier section were terminal emulation programs that gave you access to remote computers through the Internet, but that's not all a terminal emulation program can do. Before TCP/IP was ubiquitous as a way for computers to connect to networks, connectivity was handled through modems attached to the serial ports of computers. Back then, many users connected to bulletin boards (BBSes) and used menu-based systems to post messages on discussion boards, download files, and send mail to other users of the same BBS.

Nowadays, serial ports are used mainly to connect to some of peripheral devices of your computer. In microcontroller programming, they're used to exchange data between the computer and the microcontroller. For the projects in this book, you'll find that using a terminal program to connect to your serial ports is indispensable. There are several freeware and shareware terminal programs available, but to keep it simple, stick with the classics: PuTTY (version 0.59 or later) for Windows users, and the GNU screen program running in a terminal window for Mac OS X and Linux users.

Windows Serial Communication

To get started, you'll need to know the serial port name. Click Start→Run (use the Search box on Vista), type devmgmt.msc, and press Enter to launch Device Manager. If you've got a serial device such as a Wiring or Arduino board attached, you'll see a listing for Ports (COM & LPT). Under that listing, you'll see all the available serial ports. Each new Wiring or Arduino board you connect will get a new name, such as COM5, COM6, COM7, and so forth.

Once you know the name of your serial port, open PuTTY. In the Session category, set the Connection Type to Serial, and enter the name of your port in the Serial Line box, as shown in Figure 1-6. Then click the Serial category at the end of the category list, and make sure that the serial line matches your port name. Configure the serial line for 9600 baud, 8 databits, 1 stop bit, no parity, and no flow control. Then click the Open button, and a serial window will open. Anything you type in this window will be sent out the serial port, and any data that comes in the serial port will be displayed here as ASCII text.

NOTE: Unless your Arduino is running a program that communicates over the serial port (and you'll learn all about that shortly), you won't get any response yet.

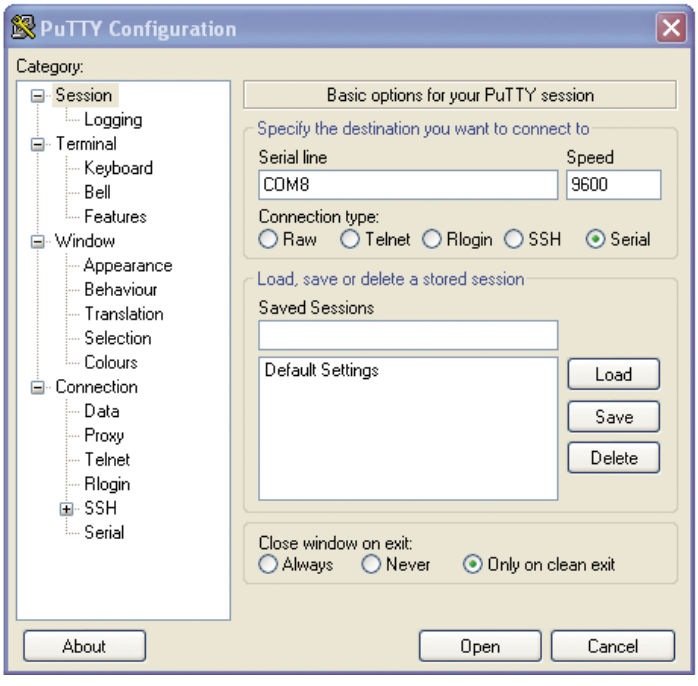
Mac OS X and Linux Serial Communication

To get started with serial communication in Mac OS X or Linux, open a terminal window and type:

```
ls /dev/tty.*    # Mac OS X
ls /dev/tty*     # Linux
```

This command will give you a list of available serial ports. The names of the serial ports in Mac OS X and Linux are more unique, but more cryptic than the COM1, COM2, and so on that Windows uses. Pick your serial port and type:

```
screen portname datarate.
```



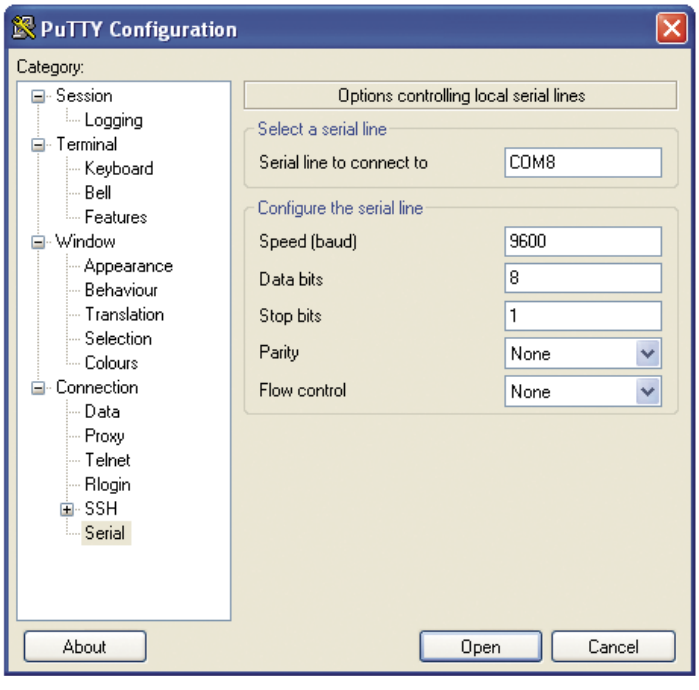


Figure 1-6
Configuring a serial connection in PuTTY.

For example, to open the serial port on an Arduino board (discussed shortly) at 9600 bits per second, you might type screen /dev/tty.usbserial-1B1 9600 on Mac OS X. On Linux, the command might be screen /dev/ttyUSB0 9600. The screen will be cleared, and any characters you type will be sent out the serial port you opened. They won't show up on the screen, however. Any bytes received in the serial port will be displayed in the window as characters. To close the serial port, type Control-A followed by Control-\.

In the next section, you'll use a serial communications program to communicate with a microcontroller.

Hardware

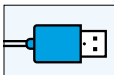
Arduino and Wiring

The main microcontroller used in this book is the Arduino module. Arduino is based on a similar module called Wiring. You should be able to use Arduino or Wiring interchangeably for the examples in this book. Both modules are the children of the Processing programming environment and the Atmel AVR family of microcontrollers. In fact, you'll find that the editors for Processing, Wiring, and Arduino look almost identical. Both programming environments are free and open source, available through hardware.processing.org. You can buy the actual modules from the original developers or from SparkFun at www.sparkfun.com or from Make at store.makezine.com. If you're a hardcore hardware geek and like to make your own printed circuit boards, you can download the plans and make your own. I recommend the former, as it's much quicker (and more reliable, for most people). Figures 1-7 and 1-8 show Wiring and several variants of Arduino.

One of the best things about Wiring and Arduino is that they are cross-platform. This is a rarity in microcontroller development environments. They work well on Mac OS X, Windows, and (with some effort) Linux.

Another good thing about these environments is that, like Processing, they can be extended. Just as you can include Java classes and methods in your Processing programs, you can include C/C++ code, written in AVR-C, in your Wiring and Arduino programs. For more on how to do this, see the Wiring and Arduino websites.

X



Who's Got the Port?

Serial ports aren't easily shared between applications. In fact, only one application can have control of a serial port at a time. If PuTTY or the screen program has the serial port open to an Arduino module, for example, the Arduino programming application can't download new code to the module. When an application tries to open a serial port, it requests exclusive control of it either by writing to a special file called a lock file or by asking the operating system to lock the file on its behalf. When it closes the serial port, it releases the lock on the serial port. Sometimes when an application crashes while it's got a serial port open, it can forget to close the serial port, with the result that no other application can open the port. When this happens, the only thing you can do to fix it is to restart the operating system, which clears all the locks (alternatively, you could wait for the operating system to figure out that the lock should be released). To avoid this problem, make sure that you close the serial port whenever you switch from one application to another. Linux and Mac OS X users should get in the habit of closing down screen with Ctrl-A Ctrl-\ every time, and Windows users should disconnect the connection in PuTTY. Otherwise, you may find yourself restarting your machine a lot.

» opposite page top

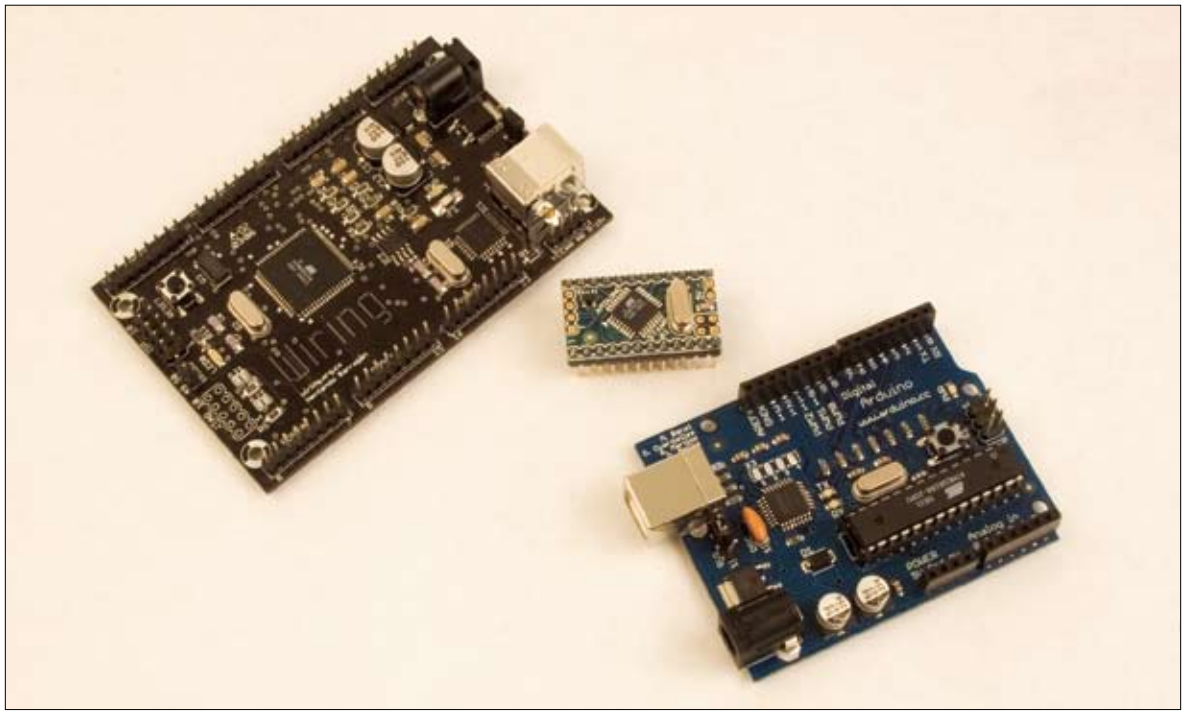
Figure 1-7

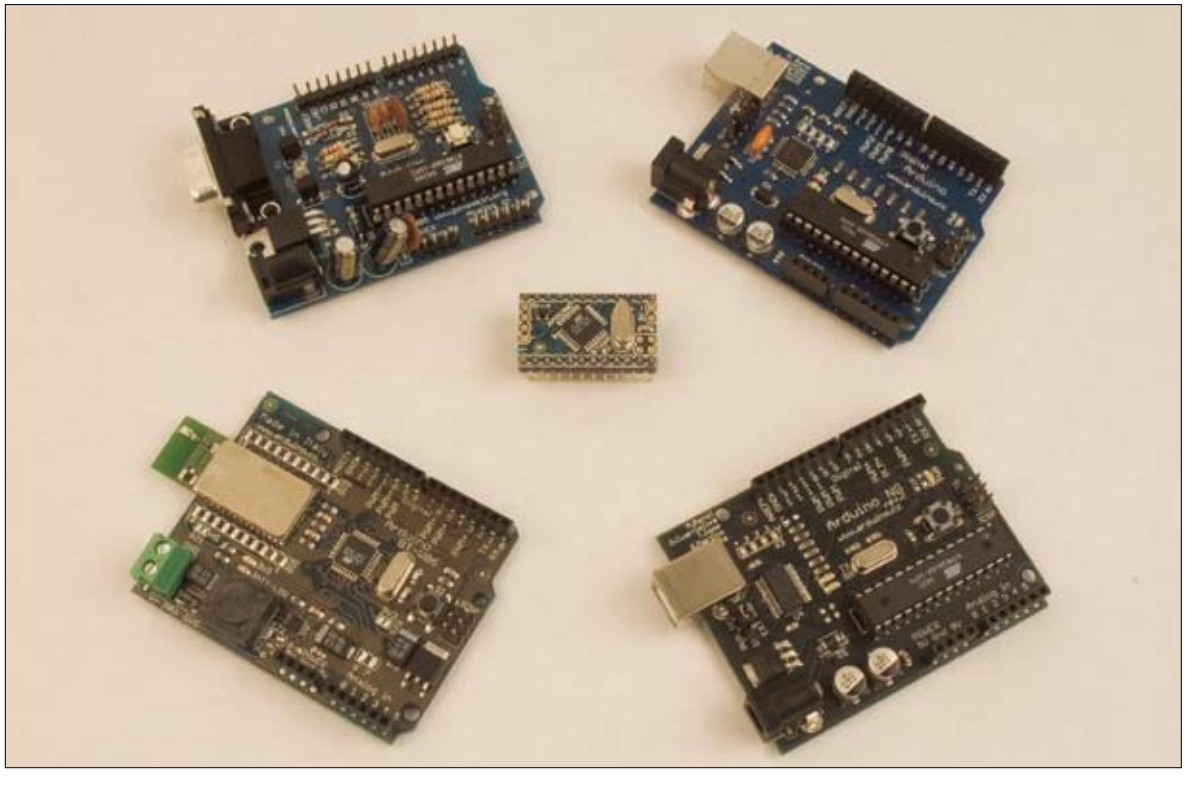
Wiring Board, Arduino NG board, Arduino Mini.

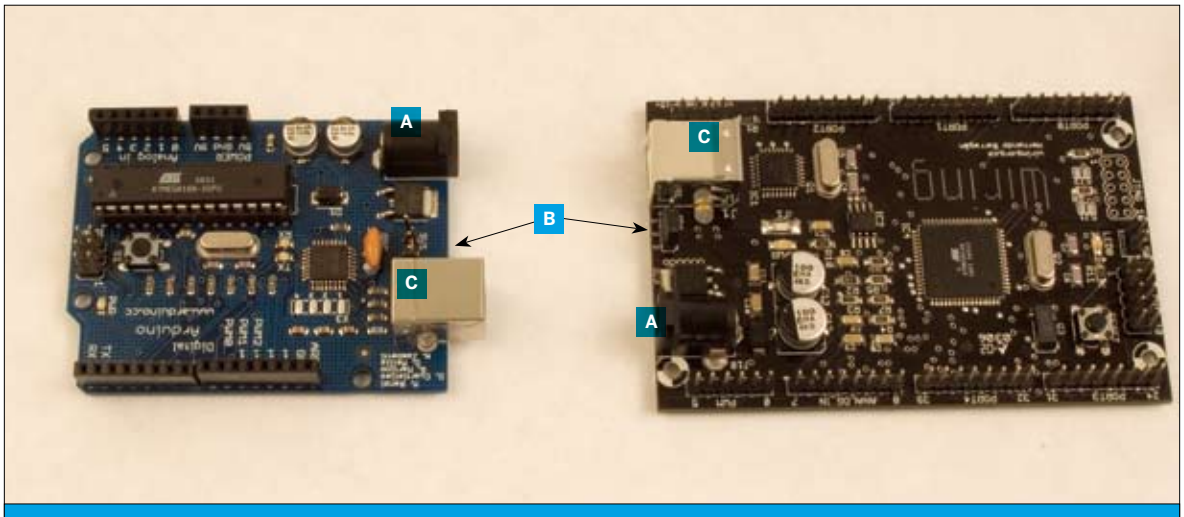
» opposite page bottom

Figure 1-8

The Arduino microcontroller modules.
CLOCKWISE FROM TOP LEFT: the original Arduino serial module; the ArduinoUSB; the Arduino NG; the Arduino Bluetooth; and finally, the Arduino Mini, center.


wiring
Arduino


Arduino



A. Power B. Power Jumper C. Serial (via USB)

Wiring and Arduino Compared

Given the similarities between Wiring and Arduino, you're probably wondering which to choose. The programming language is the same for both, and the programming environments are virtually identical, so the major factors to consider are price, size, and number of inputs and outputs.

Wiring is the larger of the two modules, and the more expensive. It has more input and output connections and some useful features such as hardware interrupt pins and two hardware serial ports. Two serial ports can be handy when you're working on projects in this book, because you can use one serial port to talk to your communications device, and another to talk to the computer on which you're programming the microcontroller. There is a software serial library for both Wiring and Arduino that allows you to use any two I/O pins as a serial port. It's more limited than a hardware serial port, in that it can't send and receive data as quickly as a hardware serial port.

Wiring boards can be ordered online from www.sparkfun.com or directly from www.wiring.org.co.

Arduino is the less expensive of the two modules, and the smaller. It has fewer inputs and outputs than Wiring, and only one hardware serial port. The Arduino developers have made a few different Arduino boards. The original board has an RS-232 serial interface, and all the components are large enough that you can solder them by hand. It was designed for people who want to make their own board from scratch. The Arduino USB board is the default board. It's not as easy to assemble by hand, but most people buy them pre-assembled. It has a USB interface. The Arduino Bluetooth board is a variant on the USB board that has a wireless interface for programming and serial communication. It's the most expensive of the Arduino models to date, but handy if you know you're going to connect to it all the time through Bluetooth. The Arduino Mini is a tiny version of the Arduino, suitable for use on a breadboard. For people familiar with the Parallax BASIC Stamp 2 or the NetMedia BX-24, the Mini is a comfortable alternative. You can also build an Arduino module on a solderless breadboard.

Arduino also features add-on modules called shields, which allow you to add pre-assembled circuits to the main module. At this writing, there are four shields on the market. PCB Europe (pcb-europe.net/catalog) sells a board for controlling DC motors, and a prototyping shield for making your own circuits. SparkFun (www.sparkfun.com) sells a breadboard prototyping shield along with the various Arduino boards. Libelium (www.libelium.com) sells a ZigBee radio shield.

The projects in this book can be built with other microcontrollers as well. Like all microcontrollers, the Arduino and Wiring modules are just small computers. Like every computer, they have inputs, outputs, a power supply, and a communications port to connect to other devices. You can power these modules either through a separate power supply or through the USB connection to your computer. The jumper shown in Figure 1-9 switches power from the external supply to the USB supply. For this introduction, you'll power the module from the USB connection. For many projects, you'll want to disconnect them from the computer once you're finished programming them. To do this, you'll need to switch the power jumper to power the board from the external power supply.

Both Wiring and Arduino have four power pins. On the Wiring board, they're labeled 5V, Gnd, GND and 9-15V. On the Arduino, they're labeled 5V, Gnd, Gnd, and 9V. In both cases, the 5V connection outputs 5V relative to the two ground pins. The 9V or 9-15V pin is connected directly to the voltage input on the external power jack, so the output voltage of that pin is equal to whatever your input voltage is. You can also use this connection to connect these modules directly to 9-15V battery power, if you set the power jumper to external power.

Figure 1-10 shows the inputs and outputs for the Arduino, the Arduino Mini, and the Wiring module. Each module has the same standard features that most microcontrollers have: analog inputs, digital inputs and outputs, and power and ground connections. Some of the I/O pins can also be used for serial communication. The Wiring and Arduino boards also have a USB connector, a programming header to allow you to reprogram the firmware (you'll never do that in this book), and a reset button. The Arduino Mini does not have these features, but they can be added using its companion USB-to-serial board. Figure 1-11 shows a typical breadboard setup for the Mini. You'll see these diagrams repeated frequently, as they are the basis for all of the microcontroller projects in the book.

Getting Started

Because the installation process for Wiring and Arduino is almost identical, I'll detail only the Arduino process here.

« opposite page

Figure 1-9

Arduino and Wiring modules. Note the jumper to switch power from the USB connection to an external power supply.

Wiring users will find things similar enough to follow along and do the same steps, substituting "Wiring" for "Arduino" in the instructions that follow.

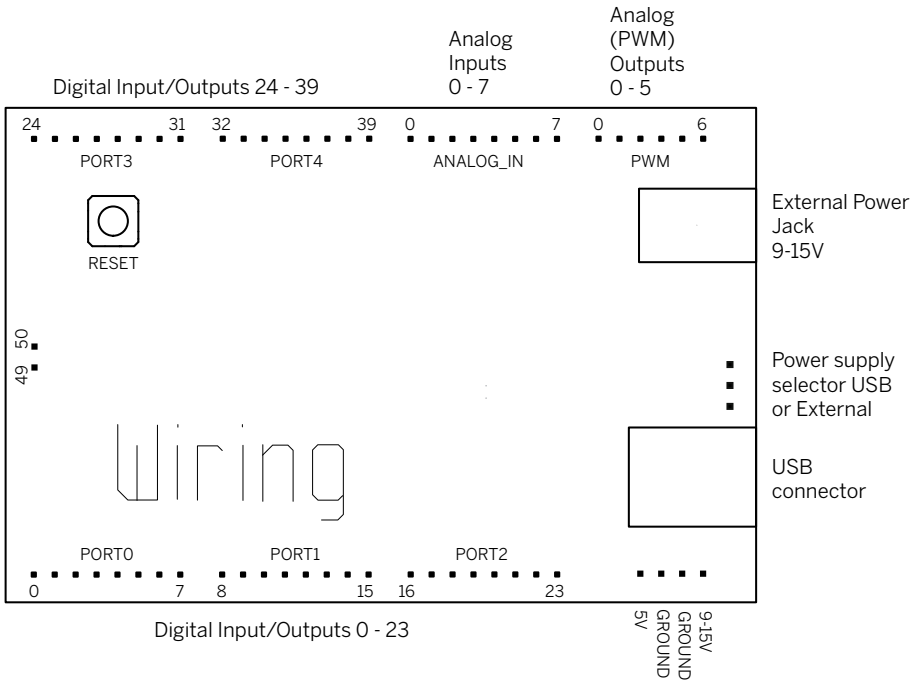
Once you've downloaded the Arduino software, you'll need to do a bit of configuring to get things ready for use. Expand the downloaded file and you'll get a directory called **arduino-0009** (if there is a newer version of the software available, the number will be different). Move this somewhere convenient: on a Mac, you might put it in your **Applications** directory; on Windows, maybe in **C:\Program Files**; on Linux, you might want to keep it in your home directory or drop it into **/usr/local**. Now navigate to the directory **arduino-009/drivers** subdirectory. In that directory, you'll find an installer for the FTDI USB serial driver (not needed under Linux). This is the USB device on the module that allows your computer to communicate with the module via USB. Install it. Macintosh users will also find a file in the **arduino-0009** directory called **macosx_setup.command**. This is the same as the **macosx_setup.command** for Processing that was described earlier, so if you already ran it to configure Processing, you won't need to do it again. If you haven't, double-click the file and follow the instructions that come up.

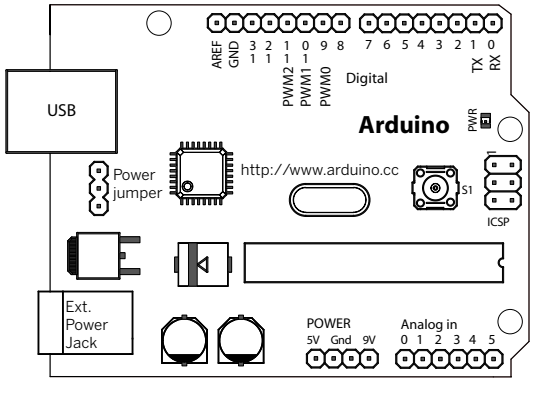
Arduino and Wiring are new to the market, and updates to their software occur frequently. The notes in this book refer to Arduino version 0009 and Wiring version 0012. By the time you read this, the specifics may be slightly different, so check the Arduino and Wiring websites for the latest details.

Now you're ready to launch Arduino. Connect the module to your USB port, and double-click the Arduino icon to launch the software. The editor looks like Figure 1-12.

The environment is based on Processing, and has the same New, Open, Save, and Export buttons on the main toolbar. In Arduino and Wiring, the Run function is called Verify. It compiles your program to check for any errors, and the Export function is called Upload to Module instead. It uploads your code to the microcontroller module. There's an additional button, the Serial Monitor, that you can use to receive serial data from the module while you're debugging.

X





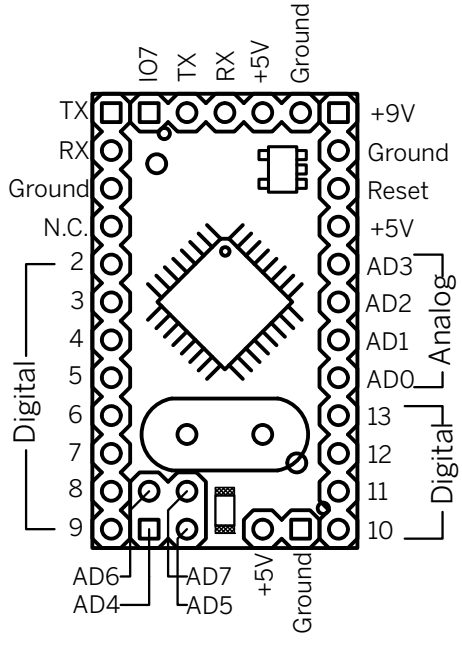
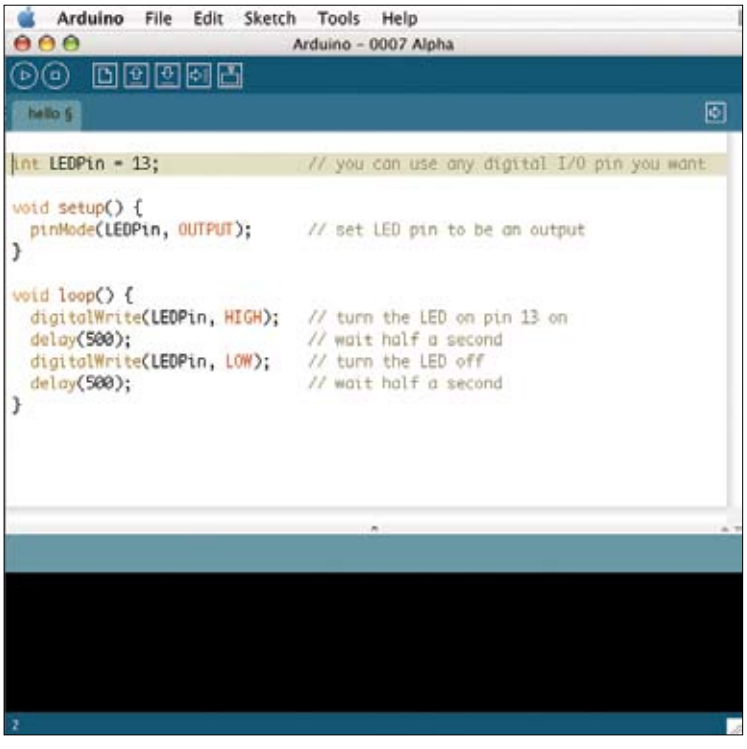


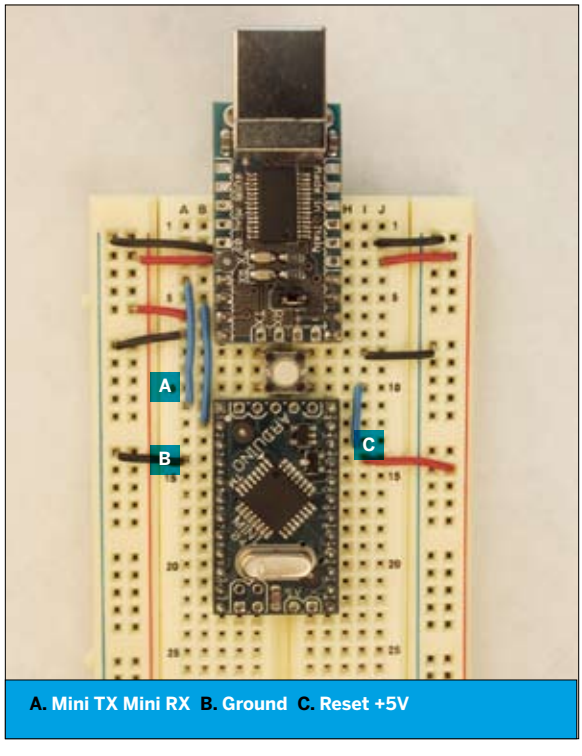
Figure 1-10
Wiring, Arduino NG, and Arduino Mini pin diagrams.

Figure 1-12
The Arduino programming environment. The Wiring environment looks identical. to this, except for the color.


Arduino File Edit Sketch Tools Help
Arduino - 0007 Alpha
hello §
int LEDPin = 13; // you can use any digital I/O pin you want
void setup() {
pinMode(LEDPin, OUTPUT); // set LED pin to be an output
}
void loop() {
digitalWrite(LEDPin, HIGH); // turn the LED on pin 13 on
delay(500); // wait half a second
digitalWrite(LEDPin, LOW); // turn the LED off
delay(500); // wait half a second
}

» bottom left

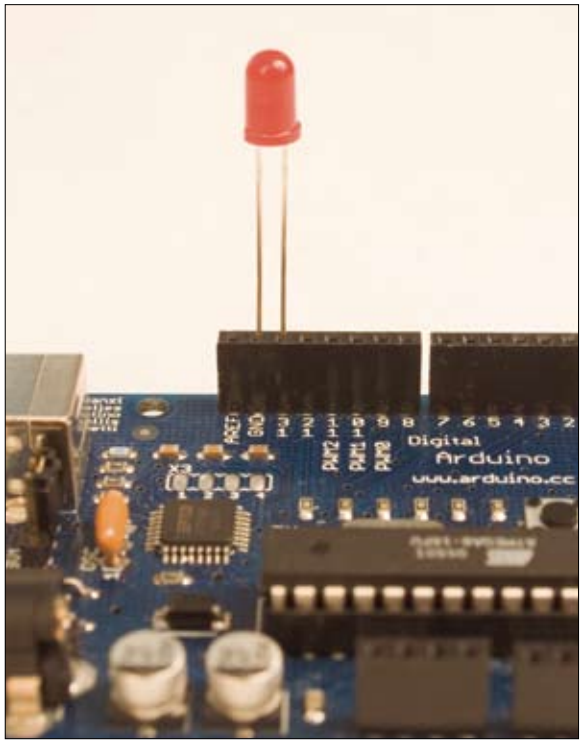
Figure 1-11
Typical wiring for an Arduino Mini.



A. Mini TX Mini RX B. Ground C. Reset +5V

» bottom right

Figure 1-13
LED connected to pin 13 of an Arduino board .



Try It

Here's your first program:

```
/*  Blink
    Language: Arduino/Wiring

    Blinks an LED attached to pin 13 every half second.

    Connections:
        Pin 13:  + leg of an LED (- leg goes to ground)
*/

int LEDPin = 13;

void setup() {
  pinMode(LEDPin, OUTPUT);    // set pin 13 to be an output
}

void loop() {
  digitalWrite(LEDPin, HIGH);       // turn the LED on pin 13 on
  delay(500);                       // wait half a second
  digitalWrite(LEDPin, LOW);        // turn the LED off
  delay(500);                       // wait half a second
}
```

In order to see this run, you'll need to connect an LED from pin 13 of the board to ground (GND) as shown in Figure 1-13. The positive (long) end of the LED should go to 13, and the short end to ground.

Then type the code into the editor. Click on Tools→Serial Port to choose the serial port of the Arduino module. On the Mac or Linux, the serial port will have a name like /dev/tty.usbserial-1B1 (the letters and numbers after the dash will be slightly different each time you connect it). On Windows, it should be COMx, where x is some number (for example, COM5).

Next, select the model of AVR microcontroller on your Arduino or Wiring module (you'll have to inspect the board to determine this). It will be either ATmega8 or ATmega168. Make the appropriate choice from the Tools→Microcontroller (MCU) menu.

NOTE: On Windows, COM1–COM4 are generally reserved for built-in serial ports, whether or not your computer has them.

Once you've selected the port and model, click Verify to compile your code. When it's compiled, you'll get a message at the bottom of the window saying Done compiling. Then press the reset button on the module to reset it and prepare it to accept a new program. Then click Upload. This will take several seconds. Once it's done, you'll get a message saying Done uploading, and a confirmation message in the serial monitor window that says:

```
Atmel AVR ATmega168 is found.
Uploading: flash
```

NOTE: If your Arduino uses an ATmega8, it will report that instead. You must make sure that you have configured the Arduino to use the model of ATmega microcontroller on your board.

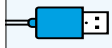
Press the reset button on the module again, and after about five seconds, the LED you wired to the output pin will begin to blink. That's the microcontroller equivalent of "Hello World!" (If you're using an Arduino Diecimila or later model, you won't have to press the reset button when you upload.)

NOTE: If it doesn't work, you might want to seek out some external help. The Arduino (www.arduino.cc/cgi-bin/yabb2/YaBB.pl) and Wiring (wiring.org.co/cgi-bin/yabb/YaBB.pl) forums are full of helpful people who love to hack these sort of things.

Serial Communication

One of the most frequent tasks you'll use a microcontroller for in this book is to communicate serially with another device, either to send sensor readings over a network or to receive commands to control motors, lights, or other outputs from the microcontroller. Regardless of what device you're communicating with, the commands you'll use in your microcontroller program will be the same. First you'll configure the serial connection for the right data rate. Then you'll read bytes in, write bytes out, or both, depending on what device you're talking to, and how the conversation is structured.

NOTE: If you've got experience with the Basic Stamp or PicBasic Pro, you will find Arduino serial communications a bit different than what you are used to. In PBasic and PicBasic Pro, the serial pins and the data rate are defined each time you send a message. In Wiring and Arduino, the serial pins are unchangeable, and the data rate is set at the beginning of the program. This way is a bit less flexible than the PBasic way, but there are some advantages, as you'll see shortly.



Where's My Serial Port?

The USB serial port that's associated with the Arduino or Wiring module is actually a software driver that loads every time you plug in the module. When you unplug, the serial driver deactivates and the serial port will disappear from the list of available ports. You might also notice that the port name changes when you unplug and plug in the module. On Windows machines, you may get a new COM number. On Macs, you'll get a different alphanumeric code at the end of the port name.

Never unplug a USB serial device when you've got its serial port open; you must exit the Wiring or Arduino software environment before you unplug anything. Otherwise, you're sure to crash the application, and possibly the whole operating system, depending on how well-behaved the software driver is.

Try It

This next Arduino/Wiring program listens for incoming serial data. It adds one to whatever serial value it receives, and sends the result back out. It also blinks an LED on pin regularly, on the same pin as the last example, to let you know that it's still working:

```
/*
    Simple Serial
    Language: Arduino/Wiring
    Listens for an incoming serial byte, adds one to the byte
    and sends the result back out serially.
    Also blinks an LED on pin 13 every half second.
 */

int LEDPin = 13;              // you can use any digital I/O pin you want
int inByte = 0;               // variable to hold incoming serial data
long blinkTimer = 0;          // keeps track of how long since the LED
                              // was last turned off
int blinkInterval = 1000;     // a full second from on to off to on again

void setup() {
  pinMode(LEDPin, OUTPUT);     // set pin 13 to be an output
  Serial.begin(9600);          // configure the serial port for 9600 bps
                               // data rate.
}

void loop() {
  // if there are any incoming serial bytes available to read:
  if (Serial.available() > 0) {
    // then read the first available byte:
    inByte = Serial.read();
    // and add one to it, then send the result out:
```

Continued from previous page.

```
    Serial.print(inByte+1, BYTE);
  }

  // Meanwhile, keep blinking the LED.
  // after a quarter of a second, turn the LED on:
  if (millis() - blinkTimer >= blinkInterval / 2) {
    digitalWrite(LEDPin, HIGH);      // turn the LED on pin 13 on
  }
  // after a half a second, turn the LED off and reset the timer:
  if (millis() - blinkTimer >= blinkInterval) {
    digitalWrite(LEDPin, LOW);       // turn the LED off
    blinkTimer = millis();           // reset the timer
  }
}
```

To send bytes from the computer to the microcontroller module, first compile and upload this program. Then click the Serial Monitor icon (the rightmost icon on the toolbar). The screen will change to look like Figure 1-14. Set the serial rate to 9600 baud.

Type any letter in the text entry box and press Enter or click Send. The module will respond with the next letter in sequence. For every character you type, the module adds one to that character's ASCII value, and sends back the result. Terminal applications represent all bytes they receive as ASCII.

Wiring Components to the Module

The Arduino and Wiring modules don't have many sockets for connections other than the I/O pins, so you'll need to keep a solderless breadboard handy to build subcircuits for your sensors and actuators (output devices). Figure 1-15 shows a standard setup for connections between the two.

Specialty Devices

You'll encounter some specialty devices as well, such as the Lantronix Xport, WiPort, and Cobox Micro. The Lantronix modules are serial-to-Ethernet modules. Their main function is to connect devices with a serial communications interface (such as all microcontrollers) to Ethernet networks. It's possible to program your own serial-to-Ethernet module directly on a microcontroller with a few spare parts, but it's a lot of work. The Lantronix modules cost more, but they're much more convenient. You'll also encounter serial-to-Bluetooth modules, serial-to-ZigBee modules, RFID modules, and other microcontrollers whose main job is to connect other devices. The details on connecting these will be explained one by one as you encounter them in the projects that follow.

Basic Circuits

There are two basic circuits that you'll use a lot in this book: digital input and analog input. If you're familiar with microcontroller development, you're already familiar with them. Any time you need to read a sensor value, you can start with one of these two. Even if you're using a custom sensor in your final object, you can use these circuits as placeholders, just to see any changing sensor values.

Digital input

A digital input to a microcontroller is nothing more than a switch. The switch is connected to voltage and to a digital input pin of the microcontroller. A high-value resistor (10 kilohms is good) connects the input pin to ground. This is called a pull-down resistor. Other electronics tutorials may connect the switch to ground and the resistor to voltage. In that case, you'd call the resistor a pull-up resistor. Pull-up and pull-down resistors provide a reference to power (pull-up) and ground (pull-down) for digital input pins. When a switch is wired as shown in Figure 1-16, closing the switch sets the input pin high. Wired the other way: closing the switch sets the input pin low.

The circuit in Figure 1-17 is called a voltage divider. The variable resistor and the fixed resistor divide the voltage between them. The ratio of the resistors' values deter-

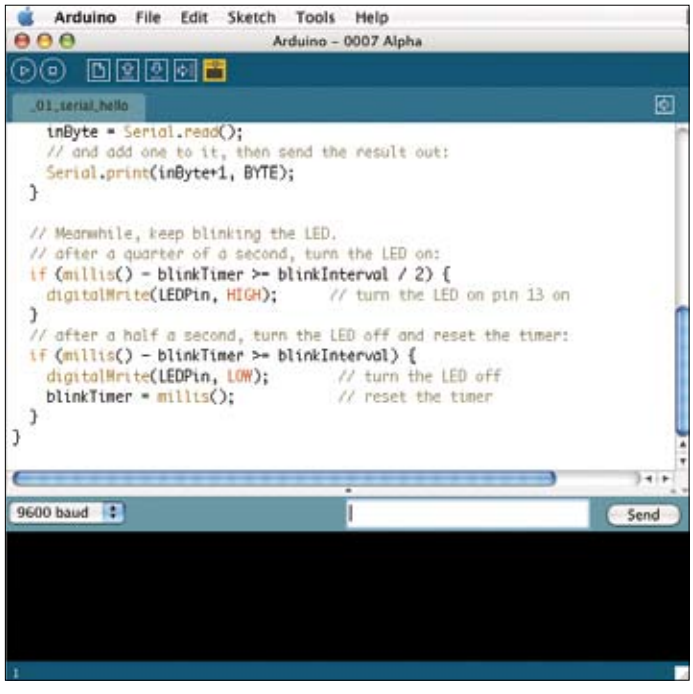

Arduino File Edit Sketch Tools Help
Arduino – 0007 Alpha
01_serial_hello
inByte = Serial.read();
// and add one to it, then send the result out:
Serial.print(inByte+1, BYTE);
}
// Meanwhile, keep blinking the LED.
// after a quarter of a second, turn the LED on:
if (millis() - blinkTimer >= blinkInterval / 2) {
digitalWrite(LEDPin, HIGH); // turn the LED on pin 13 on
}
// after a half a second, turn the LED off and reset the timer:
if (millis() - blinkTimer >= blinkInterval) {
digitalWrite(LEDPin, LOW); // turn the LED off
blinkTimer = millis(); // reset the timer
}
}
9600 baud
Send

Figure 1-14
The Serial monitor in Arduino.

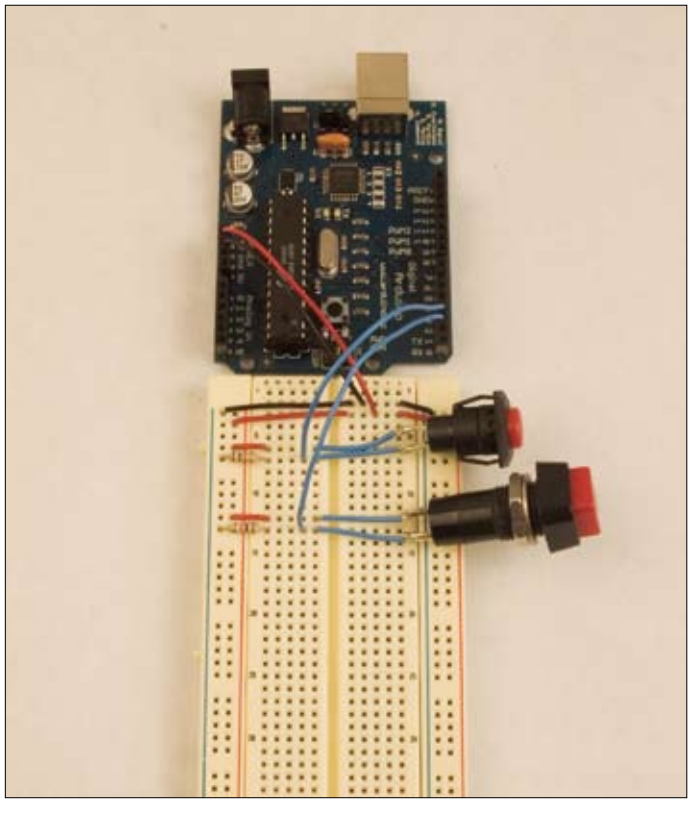


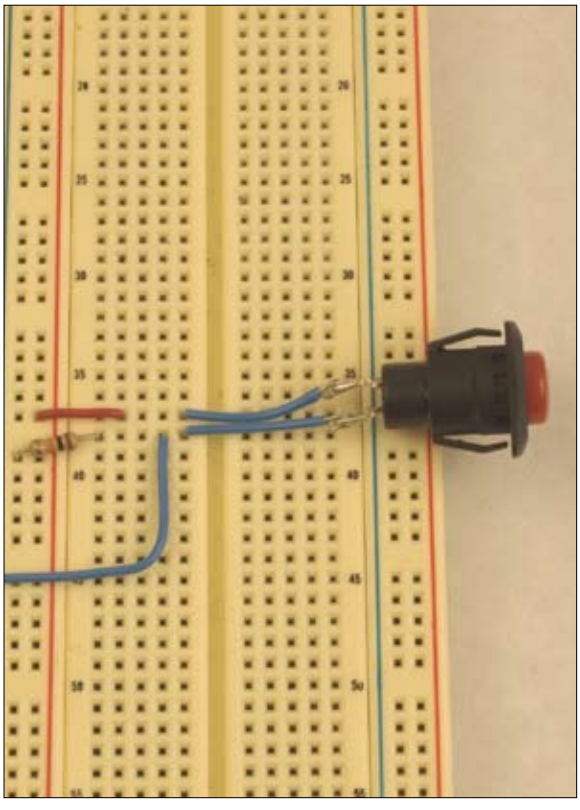
Figure 1-15
Arduino connected to a breadboard. +5V and ground run from the module to the long rows of the board. This way, all sensors and actuators can share the +5V and ground connections of the board. Control or signal connections from each sensor or actuator run to the appropriate I/O pins. In this example, two pushbuttons are attached to digital pins 2 and 3 as digital inputs.

mines the voltage at the connection between them. If you connect the analog-to-digital converter of a microcontroller to this point, you'll see a changing voltage as the variable resistor changes. You can use any kind of variable resistor: photocells, thermistors, force-sensing resistors, flex-sensing resistors, and more.

The potentiometer, shown in Figure 1-18, is a special type of variable resistor. It's a fixed resistor with a wiper that slides along the conductive surface of the resistor. The resistance changes between the wiper and both ends of the resistor as you move the wiper. Basically, a potentiometer (pot for short) is two variable resistors in one package. If you connect the ends to voltage and ground, you can read a changing voltage at the wiper.

Most of the circuits in this book will be shown on a breadboard. By default, the two side rows on each side of the board will be used for power and ground lines, typically +5V for power. On most of the boards, you'll notice wires connecting each of the side rows to two of the top rows. For some projects, the board will be powered from a Wiring or Arduino module or USB power, so there will be no need for a voltage regulator. For others, you will need one. I use separate wires rather than connecting from one side to the other directly, so that when I need a voltage regulator, it can be added easily. Figure 1-19 shows a board with and without a regulator.

There are many other circuits you'll learn in the projects that follow, but these are the staples of all the projects.
X



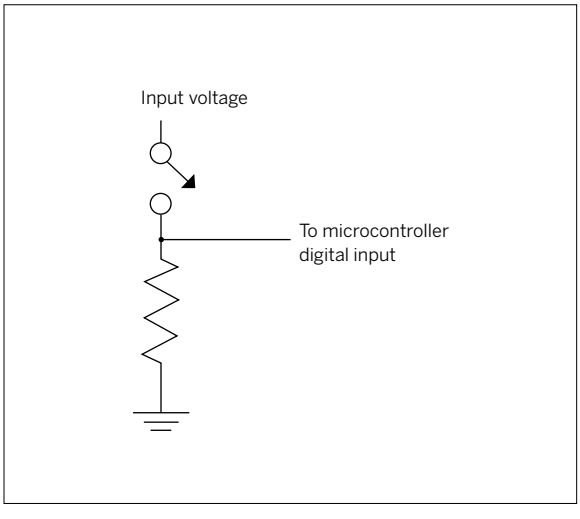
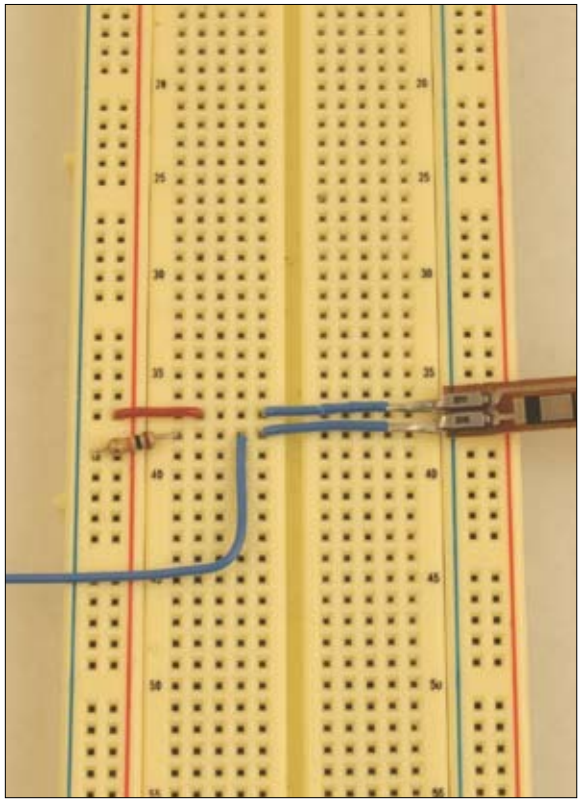


Figure 1-16
Digital input to a microcontroller.



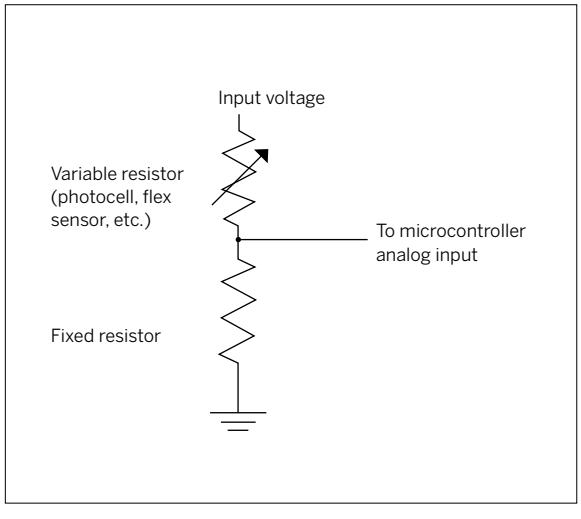
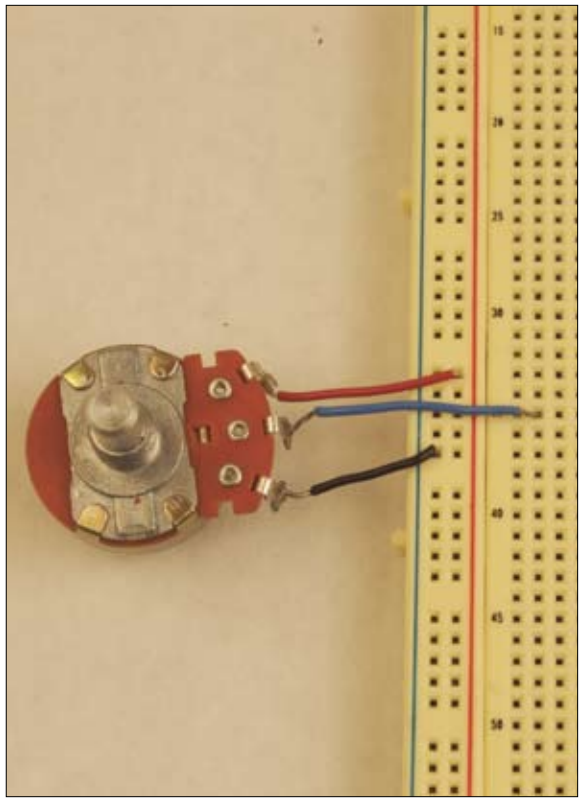


Figure 1-17
Voltage divider used as analog input to a microcontroller.



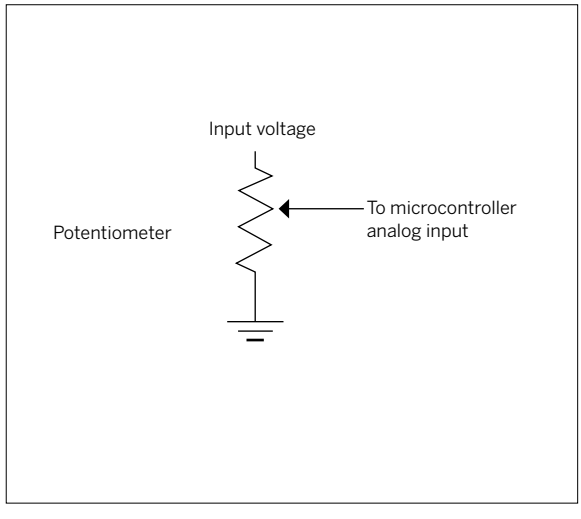


Figure 1-18
Potentiometer used as analog input to a microcontroller.

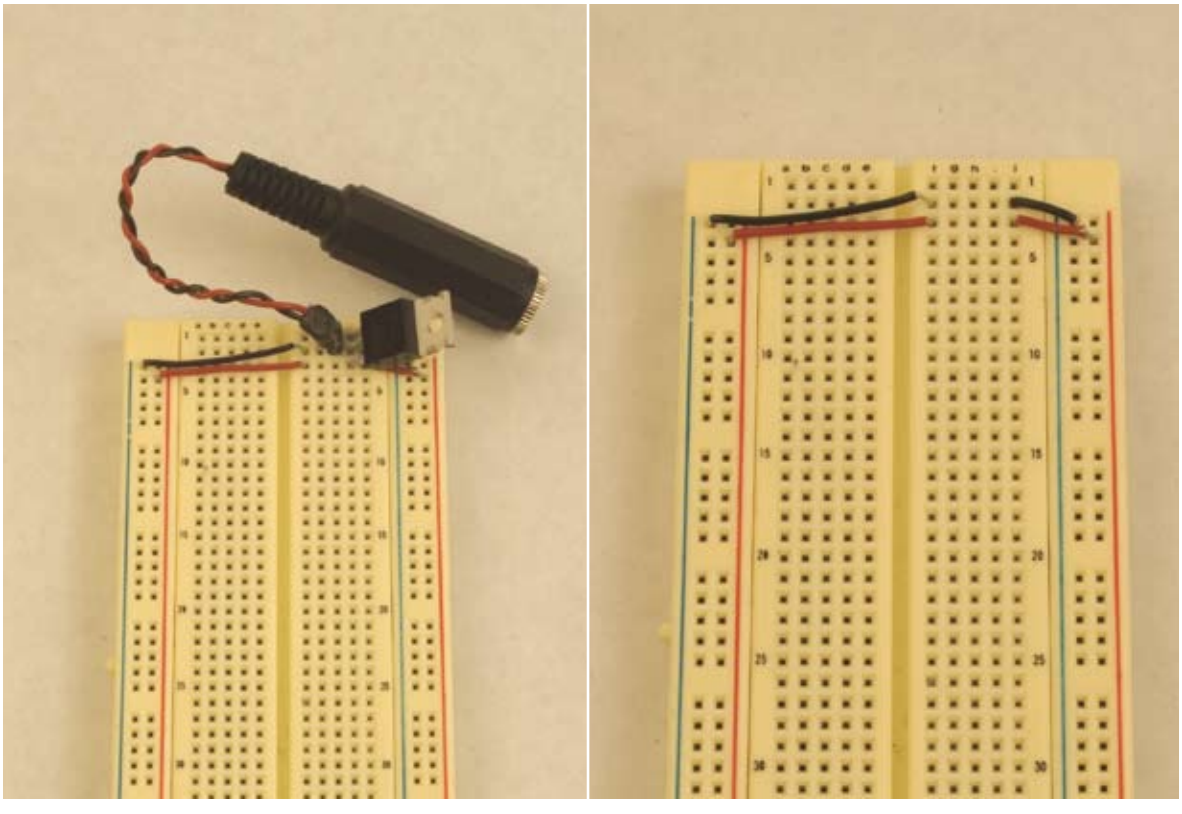


Figure 1-19
Breadboard with a regulator, and without one.

You will run across different variations on many of the modules and components used in this book. For example, the Arduino module has at least five variations, shown in Figure 1-8. The FTDI USB-to-serial module used in later chapters has at least three variations. Even the voltage regulators used in this book have different variations. Be sure to check the data sheet on whatever component or module you're using, as your version may vary from what is shown here.

It Ends with the Stuff You Touch

Though most of this book is about the fascinating world of making things talk to each other, it's important to remember that you're most likely building your project for the enjoyment of someone who doesn't care about the technical details under the hood.

Even if you're building it only for yourself, you don't want to have to fix it all the time. All that matters to the person using your system are the parts that she can see, hear, and touch. All the inner details are irrelevant if the physical interface doesn't work. So don't spend all of your time focusing on the communication between devices and leave out the communication with people. In fact, it's best to think about the specifics of what the person does and sees first.

There are a number of details that are easy to overlook, but are very important to humans. For example, many network communications can take several seconds or more. In a screen-based operating system, progress bars acknowledge a person's input and keep her informed as to the progress of the task. Physical objects don't have progress bars, but they should incorporate some indicator as to what they're doing — perhaps as simple as an LED that gently pulses while the network transfer's happening, or a tune that plays.

Find your own solution, but make sure you give some physical indication as to the invisible activities of your objects.

Don't forget the basic elements, either. Build in a power switch or a reset button. Don't forget a power indicator. Design the shape of the object so that it's clear which end is up. Make your physical controls clearly visible and easy to operate. Plan the sequence of actions you expect a person to take, and lay out the physical affordances for those actions in a sensible sequence. You can't tell people what to think about your object — you can only show them how to interact with it through its physical form. There may be times when you violate convention in the way you design your controls, perhaps in order to create a challenging game, or to make the object seem more "magical," but make sure you're doing it intentionally. Always think about the participant's expectations first.

By including the person's behavior in your system planning, you solve some problems that are computationally difficult, but easy for human intelligence to solve. Ultimately, the best reason to make things talk to each other is to give people more reasons to talk to each other.

X